NTNU

Norwegian University of
Science and Technology

Access Control in Multi-Thousand-Machine Datacenters

Håvard Husevåg Garnes

Master of Science in Communication Technology
Submission date: June 2008
Supervisor: Svein Johan Knapskog, ITEM
Co-supervisor: Philip Mackenzie, Google

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Company internal access-control systems are motivated both by internal and external policies and regulations. The American Sarbanes-Oxley act and PCI-regulations, as well as company internal privacy policies all have serious economic consequences if they are not followed. Accidental deletion of files, the misuse of logs and accidental or intentional denial-of-service attacks from internal systems will also have huge consequences for a company if they are allowed to disrupt the company value chain.

In a multi-thousand-machine datacenter, the problem of access control needs to scale beyond the normal environment of single machines and company-internal centralised login systems. The access-control system in such a computer center needs to handle issues connected with distributed systems, such as scaling beyond a normal centralised trust server, performance of communications, the need for fault tolerance and the need for clarity in the process of administering and controlling access.

The masters thesis will look into the problem of access control in a multi-thousand-machine datacenter with a mix of trusted and untrusted users, machines and applications, study existing solutions for such systems, and if found set forth new ideas for dealing with the aforementioned issues as well as other issues found during the study.

Assignment given: 15. January 2008
Supervisor: Svein Johan Knapskog, ITEM

Abstract

Large data centers are used for large-scale high-performance tasks that often includes processing and handling sensitive information. It is therefore important to have access control systems that are able to function in large-scale data centers.

This thesis looks into existing solutions for the authentication step of access control in large data centers, and analyses how two authentication systems, Kerberos and PKI, will perform when employed on a larger scale, beyond what is normal in a large data center today. The emphasis in the analysis is on possible bottlenecks in the system, computational power spent on access control routines, procedures for administration and key distribution and availability of extension features needed in large scale data center scenarios.

Our administration analysis will propose and present possible methods for initial key distribution to new machines in the data center, as well as methods for enrolling new users. We will also propose a method for automatic service instantiation in Kerberos and present a method for service instantiation in PKI. We will look at how the systems handle failed machines in the network, and look at how the systems handle breaches of trusted components.

Our performance analysis will show that under given assumptions, both Kerberos and PKI will handle the average load in a hypothetical data center consisting of 100 000 machines and 1 000 users. We will also see that under an assumed peak load, Kerberos will be able to handle 10 000 service requests in under 1 second, whereas the PKI solution would need at least 15 seconds to handle the same number of requests using recommended public key sizes. This means that some programs may need special configurations to work in a PKI system under high load.

Preface

This thesis concludes five years of studies at the Norwegian University of Science and technology, and is a full time project over 20 weeks. It has been very interesting to be able to dive into the matter of access control, and authentication in particular, over an extended period of time.

I would like to thank my professor, Svein Knapskog, and my supervisor at Google, Philip Mackenzie, for their invaluable help during the writing of this thesis. I would also like to thank the people at the Google Trondheim office for welcoming me and supporting me, especially Knut Magne Risvik for providing me with a desk, Harald Alvestrand, Jochen Hollmann and Amund Tveit for reading the thesis and giving me helpful comments along the way, Lara Rennie for help with the English language, and Roger Skjetlein for the picture on the cover.

Håvard Husevåg Garnes
10 June 2008
Trondheim, Norway

Contents

List of Tables

List of Figures

List of Abbreviations

ACL	Access control list, an object-associated list specifying subjects and permitted actions
AES	Advanced Encryption Standard
AS	Kerberos Authentication Service
ATM	Automatic Teller Machine
CA	Certification Authority, in X.509 an entity able to issue certificates
CN	Common Name, in X.509 the name under which an entity is known
CRL	Certificate Revocation List
CSR	Certificate Signing Request, an incomplete certificate sent to a CA for signing
DCE	Distributed Computing Environment
DES	Digital Encryption Standard
DH	Diffie-Hellman key exchange protocol
DoD	Department of Defence
DSA	Digital Signature Algorithm
GID	Group ID
GSI	Grid Security Infrastructure
IP	Internet Protocol
IV	Initial Vector
KDC	Key Distribution Center, common name for the Kerberos Authentication Service and Ticket Granting Service
NIST	National Institute of Standards and Technology
OCSP	Online Certificate Status Protocol
OGSA	Open Grid Services Architecture

PAKE	Password-Authenticated Key Exchange
PCI	Payment Card Industry, a set of regulations on how to handle payment card data
PIN	Personal Identification Number
PKI	Public Key Infrastructure, a hierarchy of trust maintained by signed certificates
PKI	Public key infrastructure
RSA	(Rivest Shamir Adleman) is a public key cryptosystem
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
TLS	Transport layer security
UID	User ID

Chapter 1

Introduction

The development of computers and computer networks has come a long way since its beginning when a computer operated alone, and took its input solely from humans. When computers needed to communicate with each other, people needed to physically carry floppy disks from one computer to the other (hence the term Sneakernet [Wik08]).

As computer networks evolved and computing tasks grew, more and more computers became connected to each other. Networked computers enabled the possibility of doing multiple computing tasks in parallel. By linking together multiple computers in a room in the basement, the computers could together provide the users with greater computing resources than any one computer could provide, and the data center was born. By putting many computers at the same location, the network latency decreased, the administrative overhead was reduced, and the data center computer resources could be shared amongst the users. If computing power was running low, an extra computer could be added to the data center, increasing the total power.

But the increase of computing power with each new machine added to the pool soon came to a halt. Inter-machine resource coordination and communication became bottlenecks in the data centers. Over the years, research into technology that could run on multiple machines without creating severe bottlenecks has become more and more important [47, 42].

With the growth of the web, technology has been invented that enables networking on an even larger scale than a data center, but as part of the purpose of a data center is high performance computing, we can easily see that technology for the web does not necessarily transfer to large scale high performance data centers. Even on the web, services may be overloaded and cease to function.

Likewise, the importance of security infrastructure in data centers has been recognised in later years. Data centers hold enormous amounts of information, and are often vital to the operation of companies and institutions in most parts of the world. In companies like Google the data centers are the core of the entire business operation, doing everything from crawling the web for information to indexing, query processing and presentation of search results. Google's adver-

tising and email systems are also data center applications in need of large high performance security systems.

If security in a data center is breached, the data center may lose information, company secrets may be lost, data may become unreliable or false and computing power may be misused. It is therefore vital to any data center that it has a security system capable of preventing security breaches. In some situations, data centers are not exposed to any external network, and the security infrastructure does not need to handle scenarios concerning external break-ins, but in other scenarios a data center may be so exposed that it even allows untrusted applications to run on its machines, in which case a good security infrastructure is critical to the data center.

1.1 Focus of the thesis: goals and non-goals

This thesis focuses on scalability of access control in large data centers, and we choose to define a large data center to be a data center with 100 000 machines. 100 000 machines are more computers than exist in any publicly known data center at the time of writing, although the number is in the same order of magnitude as found in publicly known data centers [Car08, Mic08a]. We therefore feel that the matter discussed in this thesis will be practical for both current and future data centers, and may form a basis for decisions and development of tailored access control systems. The study of even larger data centers or multi-data center solutions is left for further work.

We do not define scalability further than 'to be able to handle a data center with 100 000 computers'. The reason for this is that it is very hard to define [30], and we let the analysis describe the factors important for scaling. This means that the conclusion of this thesis is a discussion about each technology's strengths an weaknesses as they work in a large data center.

We have chosen to concentrate on authentication, as authentication is the first step in access control. The other part of access control, authorisation, is a problem hinging both on administration of access and the mechanism of looking up access rights, and will only be described briefly in this thesis.

We will analyse existing access control technologies that may be used in a large scale data center. We will look at properties of the technologies in terms of administration solutions and failure handling abilities as well as performance under specified assumptions. We will propose and present possible solutions to some of the administration problems faced in the system. We will also analyse the different systems for performance under specified assumptions about the data center configuration and usage.

Because of the size of the data center, it is not the goal of this thesis to do any real life performance measurements. It is also defined to be outside the scope of this thesis to describe or develop detailed and directly implementable access control protocols. We will also not look at solutions for multiple data centers of this size. In addition, we will not look at physical security of the data center.

This seems to be the first study that looks at how existing conceptual security mechanisms for authentication scale with over 100 000 computers in a data cen-

ter. As no technology that works in such a large high-performance environment is publicly described, this thesis can not be based on comparisons to previous work, and will instead be a base analysis of calculated performance.

1.2 Access control

Access control is determining who you are (identification or authentication) and what you are allowed to do (authorisation). In real life access control is deployed in many everyday situations, like determining who is allowed to enter the driver's-room in the subway, determining who is allowed to sit at the boss's desk in the office and determining who is allowed to enter the server room in the basement. These are often only enforced by code of honour or tradition, but there are also examples of physical devices to control access. An example of such a device is a door with a lock. Only a person in possession of the key is allowed to unlock the door.

In computer systems, the access control procedure is similar. A computer system identifies an individual by one of three means, something he has (like a smart card), something he knows (a password), something he is (like a fingerprint) or a combination of the three. After this identification, the computer system determines what the identified user is allowed to do. In the rest of the thesis, access control only refers to access control in computer systems unless explicitly stated.

Company-internal access-control systems are motivated both by internal and external policies and regulations. The American Sarbanes-Oxley act [Kam08] and PCI (Payment Card Industry)-regulations [Cou], as well as company internal privacy policies all have serious economic consequences if they are not followed. Accidental deletion of files, the misuse of logs and accidental or intentional denial-of-service attacks from internal systems will also have huge consequences for a company if they are allowed to disrupt the company value chain.

In a multi-thousand-machine data center, the access control system needs to scale beyond the normal environment of single machines and company-internal centralised login and centralised trust systems. The access-control system in such a computer center needs to handle issues connected with distributed systems, such as communication performance, the need for fault tolerance and the need for clarity in the process of administering and controlling both access and hardware.

1.3 Challenges of access control in large data centers

This section will give an overview of some problems and challenges with a large scale authentication system, and some solutions we have found to these problems as a result of our analysis.

1.3.1 Administration and fault handling

The administration of authentication systems is a tough challenge. Preferably this administration should be handled from a central location, as physically moving to and operating single machines is very time consuming. The remote administration needs to happen over secure lines, which means that encryption keys needs to be established securely on each machine in the data center. In this thesis we will propose a possible solution for secure initial key distribution.

If the authentication system contains a central unit, all entities needs to be able to authenticate to this unit. We will look at how initial user enrolment may be performed. If the user database is distributed, user entries may need to be edited in several places, or in the case of a pull-service the user updates will not take place immediately. Keeping object access rights synchronised with users and tasks in the organisation is also a problem undergoing a substantial amount of research [17]. If the access control system has distributed central servers, the database of authentication information also needs to be synchronised between the servers.

All data centers also need to run programs. We will propose a method for automatic service instantiation in Kerberos, where each entity in the system shares a symmetric authentication key with the central server. We will also present a method for how programs may be identified and instantiated within the system in a public key infrastructure (PKI) where cryptographic signatures are used for authentication.

Reliability in centralised authentication is also a considerable problem in a network access control system. If the central authentication server goes down, or new capabilities can not be issued, clients will not be able to alter their existing sessions. This means that a client not issued with a token for service S is now unable to receive one, and in case of a timeout an extension of an existing token is not available. The same argument is also valid in case of a network outage, whether from server failure or network congestion or other reasons.

A backup or failover server is deployable in many situations, and is often used as a remedy in such situations. In the case of connection state being stored on the server, this backup server needs to be fully up to date with the original. In the case of a capability based system with state stored in the client only, the backup server can operate independently from the main server. This would also enable simultaneous operation of two or more central servers, a solution that scales much better than one where servers need to constantly exchange client state information. It is easy to see how the central authentication server will become a hotspot in the network, and as the size of the data center grows, the number of central login servers would need to grow at least proportionally to the number of servers. We will present failure handling methods in Kerberos and PKI, and see how the systems behave if components fail.

Finally, the authentication system in a large data center should rely on as little trust as possible. Preferably the system should work without anyone trusting anyone but the access control system. By this we mean that the individual components in the data center should need to be delegated access in order to be granted access to other users' objects.

We will present a system where a trusted service on each machine is the only trusted system outside the central servers, and where a breach of security of this trusted service is not different than a breach of machine security in any computer. In addition, a breach of the trusted service will, as little as possible, affect machines other than the machine the trusted service is running on. We shall also present how Kerberos and PKI can be used for delegation of access rights.

1.3.2 Performance

In this thesis we look at how authentication performs in a data center with 100 000 machines. This creates an extra complexity because of the size of the system, and raises a new set of questions and problems of its own. How much data needs to be stored in the central server, and how quickly it must be retrieved when it is needed, are examples of problems that may become overwhelming when the data center expands. An important problem is also what happens if parts of the security in the system is breached. It is also important that the authentication system does not require a disproportionate amount of resources as the data center grows.

We will present results showing that Kerberos and PKI do not require an overwhelming amount of resources to function in large data centers under certain assumptions. We will also show that solutions exist, both for Kerberos and PKI, that are able to withstand certain security breaches without compromising the entire data center.

Other challenges comes from the throughput and latency in the chosen solution. For example, if authentication alone requires 15 seconds, this may be acceptable in a job calculating a weather report for tomorrow, but if the job is to answer a web search, a latency of 15 seconds would be too slow to be usable. Also, for the throughput in the system, we do not want a situation where the system receives 15 seconds of work orders every second, thereby having a backlog that grows to infinity.

We will present results that show that both a Kerberos solution and a PKI solution will be able to handle a data center with 100 000 machines and 1 000 users under specific assumptions of average load. We will also present results showing that Kerberos is able to handle an assumed peak load of 10 000 requests per second, whereas the PKI solution would need 15 seconds to handle the same number of requests.

Chapter 2

Theory of access control

This chapter will explore the theoretical area of access control, and provide terminology and definitions that will be used in the rest of the thesis.

2.1 Elements of access control

The procedures and entities present in an access control environment all play different roles in the process. We use the paper from Lampson et. al. [35] as a basis to define the elements of an access control system. These are illustrated in figure 2.1.

Objects	Resources such as files, devices or processes
Service	Offers access to objects
Requests	Sent to services to request access to objects
Principal/Subject	Makes requests. The principal in a data center is often a computer or a process acting on its own or on another principal's behalf, but it can also be the user of a system
Reference monitors	Examines requests and grants or denies them based on the access control rights of the principals making the requests

We also define these terms as part of the process of access control:

Authentication	The process of proving identity
Authorisation	The decision whether or not to grant a request
Session	The time period for which the authentication is valid
Log in	When a user logs in a session is started

We also define the following notation:

$\{X\}_k$	means that X is encrypted with the key k
$\langle X \rangle_k$	means that X is signed with the key k

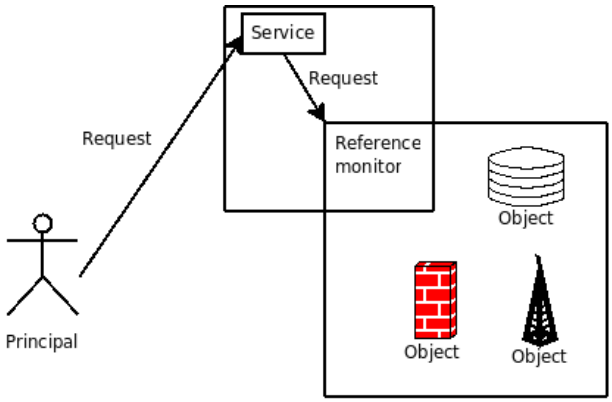

Figure 2.1: Illustration of the basic elements of access control

2.2 Protection goals

In computer security, the main goal of any security system is to achieve:

Confidentiality: That the data in the system is kept private to the authorised viewers

Integrity: That the data in the system is not changed by any unauthorised event

Availability: That the system and the data is available upon request

An access control mechanism is a factor of achieving all these goals. Unauthorised viewers are to be denied access to data, thus protecting confidentiality and integrity, and the system also needs to let authorised viewers see the data, thus providing availability. The system should also give the same protection to itself, keeping passwords, credentials and cryptographic secrets confidential and unaltered and also keeping the system available for authentication and authorisation.

2.3 Failure types in access control

There are two types of access control failures. False rejections, where a legitimate user is not able to access objects he needs to access, or false authorisation, where a non-legitimate user is allowed to access objects he should not have access to.

The default behaviour of an access control system on failure may be either to 'fail open', so that in case the system fails, access is granted, or to 'fail closed' such that a failure in the system leads to denied access. In some scenarios false rejections is the worse failure. Such systems include process control systems in a nuclear reactor or on an oil platform. In such systems, the operator's needs to be able to control the system outweighs the danger posed by illegitimate

manipulation [25]. A system that fails open might be preferable in this scenario. In most systems the false authorisation scenario is the worse. An examples of this is found where access control systems protect access to sensitive information like credit card information, personal data or military systems. In these systems a fail closed solution may be the best choice.

In this thesis we will only look at how the authentication system handles failure of components, also known as crash failures. We define that a failed machine is a machine with which we can not communicate on the network, meaning that the real failure may be either in the network or on the machine itself or both. We will also define the authentication systems' default behaviour to fail closed.

2.4 Access control overview

As described, access control involves two steps. The first step is authenticating the user, the second step is to determine what the user is and is not allowed to do. This section will describe methods for the two steps.

In some access control models these steps are performed together. An example of this is when a user presents a capability token containing embedded access rights in a model where the token is accepted without authentication. In this thesis, the two steps will be considered separate.

2.4.1 Factors of authentication

The goal of authentication is to establish the identity of a principal to the system. This can be the identity in the form of a name of a user, but it can also be the identity in the form of a role or a group, as long as the identity established is an identity the authentication system recognises and upon which the authorisation system can make authorisation decisions.

A widely used method of authentication for human subjects is passwords. The password is something that the user knows, and as long as the password is only known by the user, the password is a good authenticator.

Normally, an authentication system will use one or more of the following factors for authentication:

Something the subject knows	like a password or a pin or his mother's maiden name
Something the subject has	like a smart card or an encryption key or a certificate
Something the subject is or does	often known as biometric systems, like a fingerprint, a scan of the iris or recognition of the user's gait

If two or more of those factors are used, it is called a two-factor authentication system [48]. Examples of such systems are Automatic Teller Machines (ATMs),

where the possession of an ATM-card in conjunction with a secret PIN-number (Personal Identification Number) is used to authenticate the principal, which in this case is a human user.

Biometric authentication systems are also seeing further adoption. Fingerprint scanning is used both in personal computers and in immigration offices, but these systems are normally a lot harder to implement than a simple password system.

As this thesis looks at access control in a data center, the principals are often not human. The access control factor we will be looking at is therefore not biometric or otherwise human dependent. In the rest of this thesis the authentication factors will be referred to as credentials without regard to the actual factors used unless specifically stated. Normally this credential will have the form of a public-private key pair, a secret key or an encrypted token. It is often of value that a user should only need to enter a password once, and start many sessions based on this single password entry. This is called single sign-on, and is often achieved through the use of a temporary credential, which is issued based on the single password entry.

2.4.2 Basic methods of authentication

The process of utilising the chosen factors in an authentication system varies a lot from system to system, but there are a few basic goals to be achieved.

- The reference monitor needs to make sure that the client submitting a request is actually in possession of the credential
- The credential should not be accessible to eavesdroppers
- User friendliness - a password or fingerprint should only need to be entered once
- The storage of the credentials, such as passwords or keys, on the server needs to be protected from retrieval by malicious users

To illustrate these goals, we examine basic password authentication in detail. Many password authentication schemes employ one-way hash functions. The hash function maps an arbitrarily long string p to a fixed-length string h. Often a salt s is employed in order to make the mapping from p to h differ from system to system. Mathematically this hash function H is defined so that

$$\begin{aligned} p &= \{0,1\}^* \\ s &= \{0,1\}^m \\ h &= \{0,1\}^n \\ H : p \times s &\rightarrow h \end{aligned}$$

and also so that calculating $H(p,s)$ is an easy computational task, but finding $H^{-1}(h)$ is computationally hard. An example of a hash-function family is the set of SHA-functions from NIST [44].

Algorithm 1 Local UNIX login

1. The principal enters username u and password p
2. The reference monitor retrieves h_u and s_u from the system password file
3. The reference monitor calculates $h = H(p,\, s_u)$
4. The principal is authenticated to the system as u if $h = h_u$

Algorithm 2 Challenge-response authentication

1. The principal sends its username u to the server
2. The reference monitor retrieves h_u from the password list
3. The reference monitor issues a challenge c to the client
4. The principal responds with $R = R_1(p, c) = R_2(H(p), c)$
5. The client is authenticated if $R = R_2(h_u, c)$

An example of employment of a salted hash login scheme is from a normal local Linux login, and is shown in simplified steps shown in algorithm 1. The passwords of the users are stored in a system wide password file in the form of a one way hash of the password p and a random salt s, together with the username u of a user and the clear text salt. This way a malicious attacker is not able to read out the passwords from the password file and use them to log on, and because of the random salt involved in the computation it is a lot harder to use pre computation in the reversal of H.

This method of authentication also exists in network login systems. Telnet is a prime example of this, but there are also many other protocols that supplies the server with a clear text password over the Internet. The main problem with this approach is that an attacker can eavesdrop on the password as it travels in over an unencrypted network channel, and use it for his own authentication at a later time. Products exist that automate the process of harvesting clear text passwords, e.g. Ettercap [OV08].

Cryptographic techniques can also be used for the purpose of authentication. There are many potential ways of utilising the powers of cryptography in authentication schemes, but in all cases the principal needs to be in possession of a secret key. If this secret key is shared between the principal and the reference monitor in a symmetric cryptography scheme, the client might simply encrypt the hashed password for the reference monitor to decrypt and compare to the stored hash.

One of the earliest cryptographical methods for secure network authentication uses a process called challenge-response, and is described in algorithm 2. In a challenge-response approach the hash h of the password p is considered a shared secret between the server and the principal, and the password hash is stored on the server. The server issues a challenge c to the user, and a response function $R_1(p, c) = R_2(H(p), c)$ is utilised to calculate a response that is returned to the

server. The response R is defined such that $R = R_2(h, c)$ is computationally simple to calculate, but such that h is computationally difficult to calculate given R, c. As the server knows the shared secret, it is able to calculate the correct response on its end with the function R_2. If the response from the client matches the calculated response on the server, the authentication is complete. For an eavesdropper, knowledge of the communication does not reveal the password because it is computationally infeasible to invert R, but the reference monitor is still able to verify possession of the password for authentication.

In a public-key cryptography scheme, the principal is in possession of a secret private key, whereas the reference monitor is in possession of the corresponding public part. By signing a challenge with the private key, the successful verification of the signature of the challenge, by using the corresponding public key, is a valid proof of possession of the private key, and thereby a valid authenticator.

As the security both for public key cryptography in general and for hash functions is limited to the infeasibility of reversing the one-way function used, challenge-response authentication schemes are as secure as public key encryption techniques for authentication.

2.4.3 Methods of authorisation

Once the principal is recognised by the reference monitor, the principal can request access to controlled objects. There are two fundamental ways in which the reference monitor can decide whether or not to allow the request.

The first method is by means of an ACL, Access Control List. The ACL is a matrix associated to each object, where the rows of the matrix denote the users or groups and the columns describe the actions. The intersection between a user and an action contains the value false if the user is not allowed to perform this action on the object and true if the action is permitted.

The other method is by means of a 'capability'. This can be a forgeable or an unforgeable capability. In a forgeable capability system the principal is given a capability or token by the reference monitor, and the token itself describes the permitted actions upon presentation to the reference monitor. The token itself does not possess any proof of origin. An example of a capability is the UNIX network socket, where the program is given an integer as a return value upon opening of a socket. By presenting this integer, the program is allowed to write to the socket.

A capability can also be in the form of an unforgeable token. This token is cryptographically signed, and upon presentation of the token to a reference monitor, the reference monitor will verify the signature of the token as a proof of origin. This enables decentralised reference monitors, as the reference monitors do not need a trusted token path such as kernel memory to ensure that the token is not simply created by the principal.

2.5 Security models in access control systems

The American Department of Defence (DoD) defines two modes of access control [43].

Discretionary Access control:

> A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

Mandatory Access control:

> A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorisation (i.e. clearance) of subjects to access information of such sensitivity.

The main difference in these two modes of access control is the possibility in discretionary access control to delegate the right to access an object based on user discretion, which could technically be independent of the security policy of the system. On the other hand, in a mandatory access control system, the central security policy assigns principals a right of access the user can not further delegate or manipulate in any way.

The National Institute of Standards and Technology (NIST) defined role based access control in 1992 [17] as a supplement to mandatory and discretionary access control. In this model individual users are given a role in the system, such as teacher or officer, and each role is given privileges in the system. This way privileges can be given in the form of «teachers are able to assign grades to pupils» instead of «user 'alice' has write access to the pupils-database».

In this thesis we will mainly look at possibilities for discretionary access control. This is because discretionary access control is the main implemented access control philosophy in civilian access control systems, and also the most agile and relevant model for a company-internal data center.

2.6 Attacker models in access control systems

There are a wide range of attacker models for access control systems. In this thesis we will look at a data center with a mix of both trusted and untrusted users, applications and machines, and thus we will assume that our attacker model is the omnipresent model. In this model the attacker may have complete control over any part of the computer center, e.g. complete control over a computer or a router.

We will assume that the attacker might be able to intercept and falsify network communications, that he is able to act as a 'man in the middle', that

the attacker might at some point be the reference monitor of a connection and that all systems must be regarded as potentially hostile unless explicitly stated. However, we will not assume that an attacker is able to break any cryptography.

The chosen attacker model exposes the network to a number of conceptual attacks. Below are examples of such attacks and the consequences they may have for the operation of the data center.

An attacker may be able to read a clear text password as it travels over the network. He might also be able to intercept enough information to be able to invert H in a challenge-response scheme. He may also be able to read out session keys and other information necessary to perform an attack on the system, such as valid usernames or valid IP-addresses.

An attacker may be able to act as a 'man in the middle', and by such means intercept clear text passwords as they pass through, or present false challenges in a challenge-response scheme in such a way that H becomes invertible. The attacker may change the content of packets in such a way that re-authentication is necessary, perhaps often enough to be able to calculate secrets, or perhaps in such a way that a weaker authentication mechanism is used. In the case of predictable and repeatable communications in the authentication process, an attacker may be able to record traffic and replay it at a later stage in order to simulate the responses of the real principal.

By attacking the availability of the system, an attacker might be able to make the system use weaker authentication mechanisms, or make the user choose a less secure communication channel.

2.7 Cryptography in access control

Cryptography plays a major part of many network access control systems. The access control systems analysed in this thesis use cryptography in different ways. The two major cryptographic concepts that are used as a basis for the analysis in this thesis are symmetric-key cryptography and public-key cryptography.

Symmetric-key cryptography

Symmetric-key cryptography is a paradigm utilising a secret key for encryption and decryption. The parties in the communications all need to know the shared secret to be able to encrypt or decrypt data. Examples of symmetric-key cryptosystems is the deprecated DES (Digital encryption standard) and the current standard AES[2]. The main advantages of symmetric cryptography are that it is much quicker and less resource demanding than public-key methods, and that the keys are much shorter with the same assumed security.

Public-key cryptography

Public-key cryptography, or asymmetric cryptography, is a cryptographic paradigm utilising a trapdoor function as a means to be able to use asymmetric keys

for encryption and decryption. This means that the key needed for the encryption process is public and not equal to the key needed for decryption. The two keys are related in such a way that the private decryption key is computationally hard to derive from the public key. An example of such a cryptosystem is RSA[46].

The main advantages of public key cryptography are the possibility of publishing encryption keys and thus avoiding the need to have secrets between each possible communications partner. The concepts of public key cryptography are also possible to utilise in methods for key exchange where an eavesdropper can not derive the key from the seen information.

In addition, the possibility of using the private key as a key for electronic signatures for integrity and origin verification is a major advantage, and public key systems exist that do not facilitate encryption, but only facilitates creation of electronic signatures. DSA (Digital Signature Algorithm) is an example of such a system.

Because public key cryptography is much slower than symmetric-key cryptography, public-key cryptography is often used only as a base to agree on a symmetric key for further communications, such as in Transport Layer Security (TLS [12]).

2.8 Speed of cryptographic solutions

Both for symmetric and asymmetric cryptography, one of the main problems of using cryptography is the increased overhead on the network traffic. In the paper from Kuo [34] the time of different cryptographic approaches are measured for SSL. The setup-time for a connection using SSL is about ten times as long using RSA with a 1024-bit key as it is using 512 bit symmetric key, mostly because of public key operations.

The same paper shows that the absolute time difference between asymmetric and symmetric keys with regards to the duration of the handshake falls as the network throughput increases. However, at the same time the relative time difference increases from asymmetric keys being 3.5 times slower at 5kb/s to about 10 times slower at 100kb/s [34, figure 4.5]. As 100kb/s is much lower than normal network speed, the relative time difference in connection setup between symmetric and asymmetric cryptosystems is expected to be even greater in high speed networks.

In this thesis we use numbers from a benchmark of the Crypto++ 5.5 library [Dai08] for cryptographic analysis. In table 2.1 we see the speed of encryption with a symmetric algorithm. The speed does not change significantly with increased key length. In table 2.2 we see the speeds of the public key algorithms RSA and DSA. Encryption and signing in RSA is essentially the same operation (with the possible addition of hashing) because of the design of the RSA algorithm, and the timings are therefore almost symmetrical. We can see that doubling the RSA key-length quadruples the time of the signature and decrypt-operations, and about doubles the verification and encryption time. For DSA we notice that the signature and verification times are in the same complexity

Algorithm	MB/Second	Cycles pr. byte	µs to setup key and IV	Cycles to setup key and IV
AES/CBC (128-bit key)	84	20.9	0.431	789
AES/ECB (128-bit key)	99	17.7	0.248	454
AES/ECB (192-bit key)	86	20.2	0.242	443
AES/ECB (256-bit key)	77	22.6	0.312	572

Table 2.1: Timings of cryptography solutions in the Crypto++ 5.5 library Benchmarked on an Intel Core 2 1.83 GHz running Windows XP SP2

Algorithm	Operation	ms/operation	MCycles/operation
RSA 1024-bit key	Encrypt	0.07	0.13
	Decrypt	1.52	2.78
	Sign	1.42	2.60
	Verify	0.07	0.13
RSA 2048-bit key	Encrypt	0.15	0.28
	Decrypt	5.95	10.89
	Sign	5.95	10.89
	Verify	0.15	0.28
DSA 1024-bit key	Sign	0.47	0.85
	Verify	0.52	0.95

Table 2.2: Timings of RSA and DSA in the Crypto++ 5.5 library Benchmarked on an Intel Core 2 1.83 GHz running Windows XP SP2, taken from [Dai08]. The public exponent of RSA in this benchmark is 17.

region, both significantly shorter than a single RSA signature operation even with security-comparable keys according to [4].

RSA involves modular exponentiation with a public and private exponent for encryption and decryption, respectively. We note from [Dai08] that the encryption and signing was done on small amounts of data, and that the public exponent used in RSA was 17. This leads to short encryption and verification times, but much longer decryption and signing-times. The inverse situation, with a small private exponent and corresponding short decryption and signing times is not possible due to a lattice based cryptanalysis attack that linearises the speed of RSA cryptanalysis when the secret exponent $d < N^{0.292}$, where $N = pq$ is the modulus used in RSA [8]. There are also indications that there may be problems regarding the use of too small public exponents as well [24], but we will not describe these further in this thesis. For comparison, the benchmark from [Dai08] with public exponent 17 and a benchmark of the open source cryptographic toolkit OpenSSL [Pro08] that uses the public RSA exponent 65537 was repeated on a local machine in our laboratory and is given in appendix A. We see that the results are very similar both to each other and to [Dai08].

The Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen in Germany published a recommendation [9] in 2007 that required minimum 1024 bit RSA signature keys for security in 2007, and minimum 1280 bits for security in 2008, but recommended the use of 2048 bit keys. NIST published at the same time a recommendation of 1024 bit key length for RSA signatures [4] to be secure through 2010. In any case we must assume that a cryptographically secure system needs to utilise keys of at least 1024 bit length. The possibility of reducing key length for purposes of very short term security will be discussed later in this thesis.

2.9 Granularity of access control

The granularity of access control varies between different access control systems. In traditional POSIX systems, a user has a unique user ID, and can in addition belong to a number of groups. Objects in the system belongs to one user and one group. Each object has three sets of access restrictions, one tied to allowed operation for the owner of the object, one set tied to members of the object's group and a third set for all other users. The restrictions are divided into 'read', 'write' and 'execute', which makes the access possibilities rather restricted. It is for example not possible for an object to belong to more than one group, or to set a file append only or not accessible on Sundays. In other systems such very fine-grained policies may be supported.

The granularity of access control may also depend on the chosen set of objects. The objects in a system might only be files, so that logging on gives access to files, as we see in a Microsoft Active Directory solution, and what you do with other resources may not be controlled. In other systems, objects may also be network connections, or perhaps network connections to certain addresses and ports, limiting resources further. An object could also be access to the CPU for a certain amount of time or at a certain date.

2.10 Delegation

To explain delegation, we will define two different use cases in the data center.

If a user U in a data center authenticates to a service S, and S needs to access the user's files on server F, a question of access control arises that has two main solution strategies. In the first solution, the server F trusts the service S, and as S logs into F as S, F grants S access to all the files it controls, and trusts S to give to U only the files belonging to U. However, F does not know which user S is acting on behalf of, so if S asks for the files of user V, F will still comply and hand out the files. This scenario leads to the first use case:

1. Single principal case. A user utilises the data center by logging in to a service, and is able to gain access to a number of objects in the data center, such as processor time or stored data. The user is the only one holding its own credentials.

This use case describes a standard storage-network or a simple setup with mutual trust in the data center. When a user U logs on to service A to run a job, A may need to obtain data for the user. A will either fetch the data locally, ask the user for it or log on to service B with its own credentials and thereby be authorised to access data on behalf of user U.

The second solution involves delegation of access rights. In this scenario, when S accesses server F, it needs to prove to F that it is acting on behalf of user U. If S can not prove this, it does not have access to any files on server F but its own. If the concept of delegation did not exist, U would have needed to access the files on F itself, and transfer them to S manually. Still, if the server did not need files but instead computer power, U would have needed to act as a proxy between all servers needing to cooperate on the user's behalf. In a data center based on mutual distrust, delegation is an essential feature for scaling. This leads to the second use case:

2. Delegation case. A user logs in and gets access to a number of objects in the data center, such as a calculation framework or a job monitor. Each of these objects are delegated to work on the user's behalf in the data center, and can log onto other services in the data center to utilise further services, such as a storage network, on behalf of the user, operating with the user's delegated credentials.

In this use case, a job in the data center branches out, and trust is passed on along the line. When the user U utilises service A in the data center, and A needs to start jobs with services B and C to gain partial results or for other reasons, no service B or C will trust the credentials of service A only in order to give out data authorised only for user U, but service A will need to prove to B and C that it is in fact operating on behalf of user U.

The principle of delegation is illustrated generally in [26], where Gasser and McDermott introduce the «speaks for» and «is» notation. Formally, the notation works as follows. $\boxed{CA \mid A \text{ is } U}$ states that the CA has signed a certificate

saying that U is the owner of the key A. Further $\boxed{A \mid S \text{ for } U}$ means that the key A signed a certificate saying that S speaks for U, but is in fact not U. Together the two certificates say «The CA says that U owns the key A», and «A says that S is allowed to speak for U». When S presents the two certificates to F, F has a complete chain of information to deduce securely that S really speaks for U as long as F is given the certificate CA out of band.

Ding et.al. [14] defines several roles in a delegation scheme, somewhat modified in the list below to suit our notation. These are also explained in figure 2.2.

Delegating principal The principal whose access rights is needed to get access to the desired objects. This is the starting point of the delegation, or user U in our example.

Intermediary principal Any principal that receives a delegation to act for someone else, in our example this is server S.

Executor of the delegation The final intermediary principal. In our example this is also server S.

End point/reference monitor The enforcer of the delegation, or the reference monitor. In our example this is server F.

Ding et.al. classifies several types of delegation in [14]. Key-based and identity-based delegation is different such that in key-based delegation, whoever has the secret key can act on behalf of the owner of the key, while with identity-based delegation the final service knows that the principal is not the owner of the object, but still knows that he is acting on behalf of someone else.

Furthermore, traceable and untraceable delegation are distinguished such that in untraceable delegation, the reference monitor cannot identify the chain of delegation. The chain of delegation is fully known by the reference monitor in a traceable delegation scheme. Whether an intermediary principal can delegate further is a matter of the rights delegated to it, and does not affect whether the delegation is traceable or not.

Lastly, there is a separation between delegation with a known endpoint and with an unknown endpoint. This is defined as whether or not the delegating principal has specified the reference monitor or not.

There are also proposed schemes for role based delegation [50], but as we do not discuss role based authentication, role based delegation is outside the scope of this thesis.

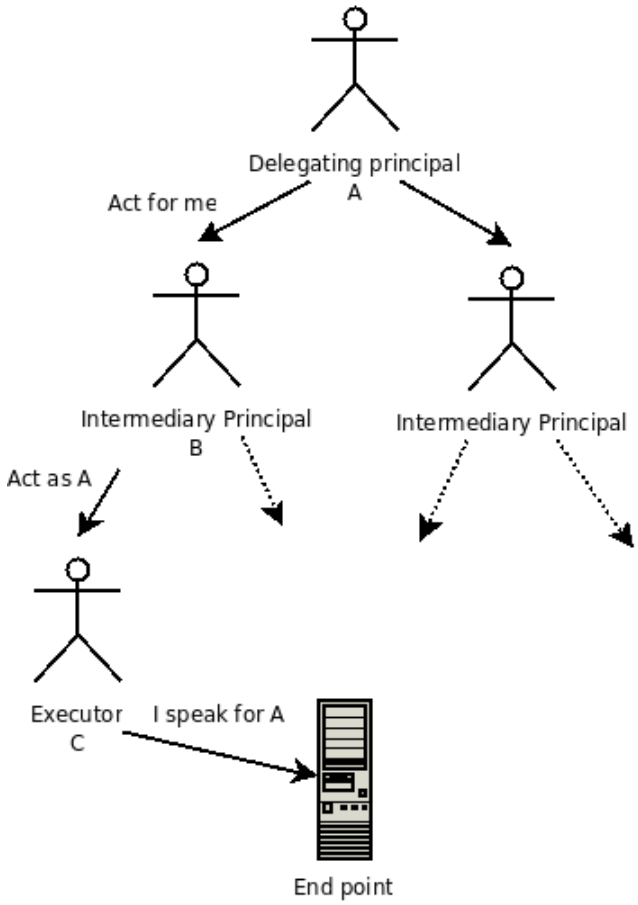

Figure 2.2: Illustration of a delegation tree

Chapter 3

Access control systems

This section will give an overview of different access control systems. The Linux solution for local authentication is described for reference purposes. Kerberos and PKI solutions are described as a reference for later analysis, and a solution for Grid systems is described for comparison purposes.

3.1 Linux login

Linux login is not a network access control system. It is described here for reference purposes and as an introduction to the conceptual workings of access control.

In Linux, the reference monitor is tightly woven into the kernel itself, and is based on the recorded UID and GID of the current process. These are stored in a data structure called *task_struct*, and there is one *task_struct* for each process. The *task_struct* is located in kernel memory, and a process can only change information in it by system calls, and all system calls are secured against unauthorised tampering with this information.

Upon all system calls and all other object accesses, such as file requests and socket request, the *task_struct* is checked to see if the user is authorised or not. As an example, the system call for opening a file is *sys_open()*. This call checks the permissions of the requested file and tries to match the group or owner of the file to the group or owner of the *task_struct* belonging to the process that called *sys_open()*. If a match exists, the permissions of the file are then checked to see if the call is allowed. If there is no match between groups or owners, the permissions for 'other' is checked in order to make an access decision.

All *task_structs* are inherited from their parent processes. On boot, the kernel will spawn the INIT-process and create the first *task_struct*. This will again spawn several other processes, which eventually prompt the user for a password. Normally in console mode, this is done via the getty and login-processes, which are both run with a UID of 0 (root[1]). As the user enters his username and

[1]The root user on a system is the highest privileged user on the system, and normally has no access restrictions

password, the functions discussed in section 2.4.2 are used to authenticate the user.

In case of successful authentication, the login program will set the UID and GID in its process to those of the logged on user, and replace itself with the user's shell. The user's shell is now running with the newly set UID and GID [WSOW08].

3.2 Kerberos login

Kerberos is one of the most widely used network access control systems today. It was developed by MIT as a part of the Athena-project to enable seamless login from any workstation on campus without the use of different passwords. The first public release of Kerberos was in the late 1980s and was version 4 of the protocol. Today the current version is version 5, which was released in 1994, and amongst other changes implemented support for public key cryptography. We will, however, consider only the symmetric key scenario of Kerberos.

Kerberos is intended for large scale access systems, and is designed for scalability [41]. Kerberos version 5 is also used as the authentication service in The Open Group's Distributed Computing Environment (DCE) system [27, 54, Gro08], with the addition of extra authorisation features that is outside the scope of this thesis.

In a Kerberos authenticated network [32], a central reference monitor, the Kerberos Authentication Service, (AS) is responsible for authenticating users, taking the responsibility for authentication of users away from the services. In a three step authentication scheme, the AS, trusted both by the service and the user, establishes trust between the two parties, as shown in figure 3.1.

The steps of the process of authentication and session initiation are as follows, simplified for readability as described in [49]. A detailed description and explanation is found in [33] and [32], but is left out of this thesis as the analysis of the protocol does not require detailed packet and message descriptions.

The concept is based on the AS providing the principal with information about the principal itself, encrypted with the secret key of the service. When the service is able to decrypt the principal-information provided by the principal, the service knows that the AS is the only possible source of the encrypted material because it is the only entity in possession of the secret key besides itself, and that the principal therefore must have been authenticated with the AS in order to have been able to obtain the information.

First, and out of band of the normal protocol, a shared secret in the form of a password is established between each user and the AS. From this password we derive a symmetric encryption key. Also we introduce a Ticket Granting Server (TGS) that is a different service to the AS, but often resides on the same machine. The TGS is an indirection service necessary to provide for single sign-on. Together the TGS and the AS are called the Key Distribution Center (KDC). The TGS and the AS also share a secret key. Further, all services register a shared secret key with the KDC.

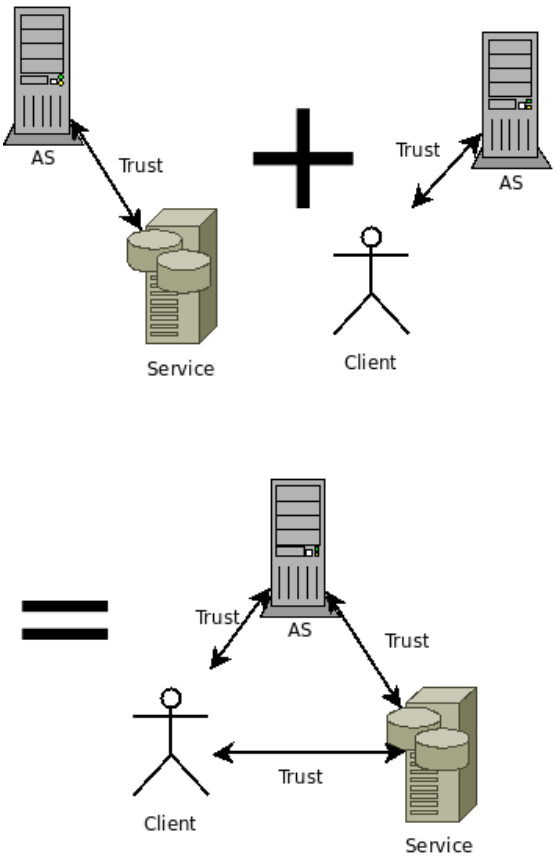

Figure 3.1: Kerberos AS establishes trust between parties

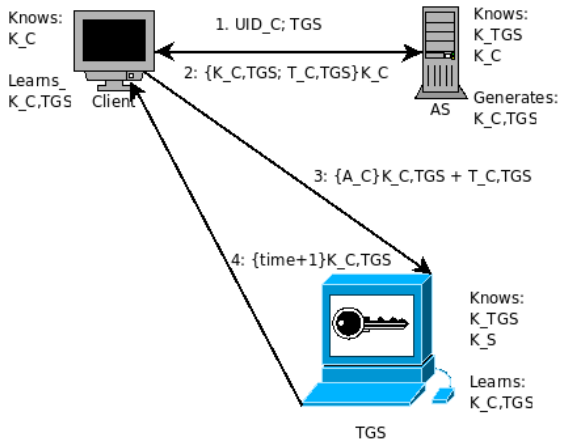

Figure 3.2: Kerberos authentication and TGT issue

The authentication works as illustrated in figure 3.2. In message 1, the client asks the AS for a ticket to use the TGS. This message consists of the username (UID_C) of the client and the name of the ticket granting service (TGS) along with possible options for the ticket, such as possibility of ticket delegation. The entire response in message 2 is encrypted with the key derived from the user's password, and consists of a randomly generated session key ($K_{C,TGS}$), and a section encrypted with the secret key between the AS and the TGS called a ticket ($T_{C,TGS}$). This ticket contains the client's name, the name of the TGS, the current time, the lifetime of the ticket, the IP of the client and the same session key, also along with possible options for the ticket. When the user enters his password, he is able to see the session key but he is not able to decrypt the contents of the ticket. This initial ticket for the TGS is called a Ticket Granting Ticket (TGT).

Now, to request the service of the TGS, the user presents it with the ticket received from the AS. Message 3 consists of one part called the authenticator (A_C) which is encrypted with the session key and contains the name and IP of the client and the current time along with a new generated sub-session key. The sub-session key is optionally used by the TGS but is used for further communication in the server authentication phase. The other part is the TGT. The TGS is able to decrypt the ticket, and then the authenticator by using the session key from the ticket, and when the information in the two parts matches, the TGS knows that the user has been able to get the ticket encrypted with a key known only to itself and the AS, and that the client therefore must have been authenticated by the AS. The timestamp has the function of a nonce (number only used once) and prevents replay attacks. If the client wants the server to authenticate itself, the server sends timestamp+1 back to the client in message 4, encrypted with the session key. This tells the client that the server was able to learn the session key by decrypting the ticket with the key known only by the service and the KDC.

When the client wants to be authenticated to another server S, he will run

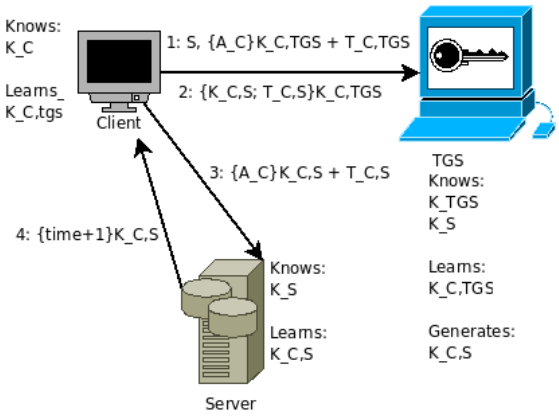

Figure 3.3: Kerberos authentication to server

through the same protocol as to get the TGT, but with two slight differences as shown in figure 3.3. The user will first send the name of the server (S), the TGT and the encrypted authenticator A_C to the TGS. With the TGT and the A_C, the TGS is able to authenticate the client. Now the TGS will generate a new session key for the session between C and S ($K_{C,S}$). It will then make a ticket for the server encrypted with the secret key of the server (K_S) and send the session key and the ticket back to the client, encrypted with the session key contained in the TGT. This step is similar to the AS authenticating the client for the TGS, but with the exception that the user's password is not involved in the authentication. This way single sign-on is achieved via TGS indirection.

In standard Kerberos, a ticket is set with 8 hours expiration time, but the principal can request this to be set to a different length. In order to have long-lived tickets, a renewable ticket may be issued with two expiration times, one expiration time for the ticket and one expiration time which is the latest time the ticket can be renewed. The ticket may be renewed within the ticket lifetime by presentation to the KDC, and the new expiration cannot exceed the renewal expiration time. The renewed ticket will have the same renewal expiration time as the original ticket, and after this time the ticket is permanently expired. Upon renewal, the KDC may check a ticket hotlist for ticket revocation purposes, and deny renewal if the ticket is listed as compromised.

3.2.1 Delegation in Kerberos

As mentioned in section 3.2, a ticket in Kerberos can contain a number of options [32]. Four of these options describes the possibility in Kerberos of doing delegation. The options are:

FORWARDABLE This option is normally only relevant for the TGS. It describes that the TGS is allowed to issue other TGTs based on this ticket, but that contain a different source IP-address. Thus, a client A can retrieve a TGT from the TGS containing the source IP of machine B. If this

ticket is given to a service on B, this service can use the ticket to request additional tickets from the TGT on A's behalf, thus in effect acting as A.

FORWARDED This option is set in a ticket as a result of it being issued by way of a FORWARDABLE-option.

PROXIABLE Like FORWARDABLE, this enables the issue of another ticket with a different IP, but not a TGT. This way the client A can give a service S_1 another service-ticket for service S_2 that enables S_1 to identify itself as A for service S_2, but without enabling S_1 to be issued a TGT and thereby preventing S_1 to access any other services on A's behalf.

PROXY This option is set in a ticket as a result of it being issued by way of a PROXIABLE-option.

3.2.2 Advantages of Kerberos

For use in a large scale data center, one of the most important features of Kerberos is that it does not store any session state on the side of the KDC. All needed session state is embedded in the tickets themselves. Also, Kerberos is a single-sign-on system. The user only needs to enter his password once per session, in order to get the TGT. After that, the TGT is used to enable further access.

This is opposed to systems like NTLM in Microsoft domains [55, Gla08, 57]. In the NTLM system, greatly popular in the late 1990s and early 2000, the central server was called a domain controller, and all clients logged on to this domain controller using the NTLM-login protocol. NTLM was able to act as a single sign-on service in a way totally unlike Kerberos. As discussed in the Microsoft TechNet Article [Tec08], an external object or service such as a printer in a domain based on NTLM needed the domain controller to verify directly the identity of the user using a Security Access Token for every access request. This is in great contrast to the Kerberos method where the service and the TGS do not speak directly. The limitations of this mean that caching of credentials is not possible, and the domain controller needs to send the Security Access Token on every use of the external object. This is shown in figure 3.4 taken from [Tec08]. Microsoft moved away from NTLM, a process that accelerated with Windows 2000, where Kerberos was the preferred protocol. NTLM is still supported in Windows for compability reasons.

With a stateless system, the server does not need to maintain any information on each client, as previously discussed in section 1.3. This makes it easier to setup failover systems and decentralised systems with multiple servers, all as long as the utilised encryption keys are the same. A decentralised TGS (TGS_2) needs to have the same shared key with the AS as TGS_1 in order for the TGT to be usable. This, however, leads to secret keys being in several places at once, and the possible increase in the probability of key compromise. Furthermore, if TGS_1's key is compromised, TGS_2 might not know and the security in the system is compromised, even if TGS_1 the key compromise is known to and acted on by TGS_1.

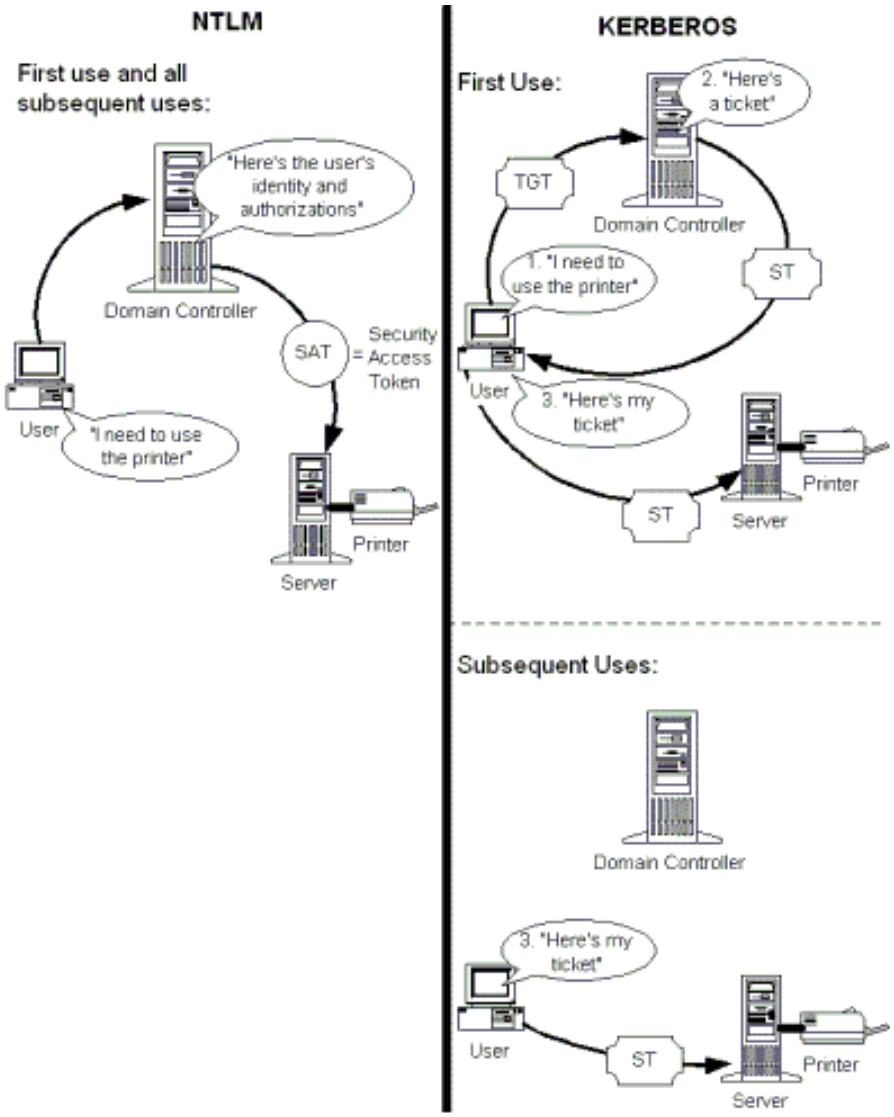

Figure 3.4: Single Sign On, the difference between NTLM and Kerberos [Tec08]

The overhead in Kerberos is relatively small. For each new authentication by a principal P to a new service S, three messages are needed. The first message, to get a service ticket, requires one encryption by P for the authenticator. The TGS decrypts the message twice: once to decrypt the authenticator and again to decrypt the TGT. Then the TGS generates one random number for the session key and encrypts this to create the ticket and encrypts again with the TGT-key to ship the ticket back. The client then needs to decrypt the session key. To get access, the client builds an authenticator that is encrypted and contains a random number for the sub-session key, and the service - as the TGS - decrypts twice to get the ticket and the authenticator. If the server needs to authenticate itself, one more encryption/decryption is required. Further in the connection the data is encrypted normally, using the sub-session key from the authenticator.

All in all this totals to the client needing to execute the crypto-algorithm 3 or 4 times and generate one random number, the TGS needing to generate one random number and run the crypto-algorithm 4 times, and the service needing to run the crypto-algorithm 2 or 3 times. As these are symmetric algorithms, they are relatively cheap compared to public key operations.

3.2.3 Advantages of Kerberos delegation

Delegation in Kerberos has the advantage that it is just as lightweight as the protocol itself, and requires no additional infrastructure. A delegation ticket is a normal ticket with attributes suitable for delegation, and is neither larger nor more complex than an ordinary ticket. Also, a server receiving a ticket does not need special routines for verifying the authentication or the delegation chain.

The use of the central server for issuing the delegation tokens means that tight central control can be held over the delegation process, enabling strict mandatory access control. Also the differentiation of proxy and forwarding tickets means that the delegation chain can be actively stopped in case of an untrusted leaf node in the tree. If this leaf node is only issued a proxy-ticket, it can only access the services the tickets are issued for, and not general services via a TGT.

3.2.4 Weaknesses of Kerberos

Kerberos is an authentication protocol, and not an access control system. Thus, the authentication is taken care of, but the authorisation needs to be handled some other way. This means that a Kerberos authentication token needs to be handled by a local trusted reference manager on the computer where the actual requested objects reside. A possible way of doing this is having a trusted login service on each computer, and using the service name in the ticket to make the login service start the named service with the privileges of the authenticated user, as it is done in local Linux login.

Harbitter et. al. [28] points to the key distribution as the major problem for Kerberos. As the subjects of Kerberos are made smaller and smaller, the number of individual shared secrets grow. Every service utilising Kerberos needs to have one shared secret with the TGS. This secret needs to be changed regularly, and the complete security of the system is dependent on this secret staying secret.

Jobs in a data center may be long-running, and if tickets expire before the job is finished, the Kerberos server may need to renew the ticket to enable job completion. Even if expiry times are randomised, this may happen in situations where tickets are requested with relatively short life-times and the job lasts longer than intended. If the ticket is passed its renew-expiration time, the ticket may need to be reissued. If the delegation tree is big, a lot of tickets may again expire around the same time, creating a bulk of near-simultaneous requests that the Kerberos server must be able to handle without breaking the running job by not reissuing tickets in time.

3.2.5 Weaknesses in Kerberos delegation

The delegation process still needs server interaction in Kerberos, which may be an obstacle to its scaling properties. Every delegation needs approval and computation on the TGS, making it quite service intensive if the delegation tree expands. For each delegating connection in the tree, two interactions with the TGS is required. In a complete n-level t-tree (a complete n-level tree where all nodes has t children), the number of network connections to the TGT will then be $\sum_{i=1}^{n} 2t^i$, showing exponential growth if $t > 1$.

3.3 PKI-systems

A Public Key Infrastructure (PKI) based on standardised certificates like X.509 is not in itself a standardised authentication system. The properties of CA-issued certificates are nevertheless usable for authentication. This section gives an overview of how the PKI standards are used in authentication systems.

3.3.1 Certificates

X.509 is a standard describing a certificate [56, 31]. A certificate is a digitally signed collection of credentials, among them a public key together with information identifying the owner of the key. The certificate is digitally signed and thereby issued by a Certificate Authority (CA). This CA confirms with its signature that the public key in the certificate belongs to the entity identified in the same certificate. An example of a X.509 certificate is given in listing 3.1.

The values noteworthy in the listing are as follows:

Serial number
: the combination of the issuer and the serial number is unique, enabling the identification of the certificate

Signature algorithm
: this describes the algorithm used to sign the certificate by the issuer

Issuer
: CN (common name) describes the name of the issuer

Subject
: CN describes the common name of the owner of the key in the certificate, the subject

Subject public key info
: this field describes the public key of the subject

X509v3 extensions
: describes other info in the certificates, like intended use of the certificate, if this certificate can be used to issue further certificates (the CA extension in listing 3.1 is set to false, and the certificate can therefore not issue further certificates) and information on where to find the certificate of the issuer and revocation information

Signature Algorithm
: the signature of the certificate, generated with the given algorithm and with the private key belonging to the issuer

3.3.2 Certificate hierarchies

As mentioned in section 3.3.1, a certificate is in all cases signed by a CA. A CA can issue certificates further able to act as CAs, making a certificate hierarchy called a certificate chain. A certificate chain is a chain of certificates, where certificate C_n is used to sign certificate C_{n+1}. The root of the chain's certificate needs to be distributed out of band to the validators, but each subsequent certificate in the chain can always be validated using the previous certificate, and need thus not be transferred out of band in order for the certificates to be verified as valid. Still, the complete chain of certificates needs to be present at the validation point of the last certificate in the chain in order for the validation to be possible.

By simple signature verification, X.509 enables a cryptographically verifiable link between a subject and the subject's public key, validity dates of this key and other properties included in the certificate.

3.3.3 Certificate issuing

A certificate may be issued in different ways. We will be looking at two possible issuing methods, the CA-generated certificate and the CSR (certificate signing request).

In the case of a CA generated certificate, the CA gathers information on the subject of the certificate, generates a key-pair, embeds the public key in the certificate together with the relevant information and sends the certificate and the private key to the subject. This has the advantage that the process is completely under the control of the CA, and that the CA is able to recover the subject's private key in case this is archived by the CA. Disadvantages of this methods are that the private key is known both by the CA and the subject, and that the private key needs to be transferred out of the machine on which it was generated. This can lead to key compromise and loss of security.

Listing 3.1: X.509 certificate for www.google.com

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            68:76:64:38:3d:49:6e:2e:f5:e3:19:98:42:e0:7c:ee
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd.,
                CN=Thawte SGC CA
        Validity
            Not Before: May  3 15:34:58 2007 GMT
            Not After : May 14 23:18:11 2008 GMT
        Subject: C=US, ST=California, L=Mountain View,
                 O=Google Inc, CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:e6:c5:c6:8d:cd:0b:a3:03:04:dc:ae:cc:c9:46:
                    be:bd:cc:9d:bc:73:34:48:fe:d3:75:64:d0:c9:c9:
                    76:27:72:0f:a9:96:1a:3b:81:f3:14:f6:ae:90:56:
                    e7:19:d2:73:68:a7:85:a4:ae:ca:24:14:30:00:ba:
                    e8:36:5d:81:73:3a:71:05:8f:b1:af:11:87:da:5c:
                    f1:3e:bf:53:51:84:6f:44:0e:b7:e8:26:d7:2f:b2:
                    6f:f2:f2:5d:df:a7:cf:8c:a5:e9:1e:6f:30:48:94:
                    21:0b:01:ad:ba:0e:71:01:0d:10:ef:bf:ee:2c:d3:
                    8d:fe:54:a8:fe:d3:97:8f:cb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication,
                TLS Web Client Authentication,
                Netscape Server Gated Crypto
            X509v3 CRL Distribution Points:
                URI:http://crl.thawte.com/ThawteSGCCA.crl
            Authority Information Access:
                OCSP - URI:http://ocsp.thawte.com
                CA Issuers - URI:http://www.thawte.com/
                                 repository/Thawte_SGC_CA.crt
            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
        93:a4:8e:05:9d:7d:8a:f3:f8:32:d0:3b:9c:21:ce:d2:e8:55:
        fd:80:b5:bb:d5:2b:54:7a:25:ac:af:73:18:0a:f9:b7:7a:99:
        5c:16:23:46:57:fc:31:19:5b:8b:f2:04:79:73:ee:b4:b2:56:
        6b:df:d7:f7:d8:56:d5:b7:aa:cd:e8:9c:c8:99:f3:76:4b:64:
        07:ad:ea:9a:2b:20:92:e6:92:9b:32:84:7c:82:62:77:9a:15:
        a0:d7:21:ad:c8:d9:8c:bb:31:82:9b:10:86:a9:41:7a:12:e0:
        01:56:09:06:d8:63:9a:50:ee:44:ad:de:75:41:01:7a:69:53:
        49:8a
```

Algorithm 3 Authentication using certificates

1. The principal presents his certificate to the reference monitor and asks to be authenticated
2. The reference monitor sends the principal a random challenge
3. The principal digitally signs the challenge using his private key
4. The reference monitor verifies possession of the private key and authenticates if the signature is verifiable by using the public key in the presented certificate

The CSR method is a method where the subject generates his own key pair and certificate and signs the certificate with the private key. Then he sends the self-signed certificate called a CSR to the CA, who verifies the signature and replaces it with its own signature and this way issues the certificate. Advantages of this method is that the private key is generated by the subject, and therefore does not need to be transferred and does not need machine resources for generation by the CA. Disadvantages include more logic at the client side to enable key and certificate generation. Both methods need authenticated channels between the subject and the CA, but the CSR channel does not need confidentiality.

3.3.4 Using certificates for authentication

The public key in the certificate is thereby possible to use in a verification procedure to prove that the private key is in the possession of the principal presenting the certificate. Assuming as with a password, that the private key is kept secret, such a verification process is proof for a reference monitor that the identity of the principal is the one given in the certificate, and thus authenticates the principal.

The verification procedure can be done in many ways, but one possible way is described by FIPS in [18], and works in a simplified form as outlined in algorithm 3.

3.3.5 Delegation in a PKI

As a certificate can be issued by any CA certificate, delegation in a PKI system can be achieved by the simple method of the delegating principle A issuing a certificate to the delegated principle B with his certificate.

By using X.509 Proxy Certificates as detailed in Welch et. al. [52], a certificate can contain many elements well suited for delegation purposes, such as the maximum length of a delegation chain, specifying that only a subset of user A's privileges are delegated and specifying whether to allow the proxy certificate to delegate privileges further. This approach also generates a new key for each delegation certificate. An illustration of this approach is shown in figure 3.5. We see in the figure that the parties first exchange certificates and authenticate one another. Then the delegate generates a CSR, authenticated with its public key,

and sends it to the principal. The principal then issues the delegation certificate by signing it with its own key. The next step in the delegation proceeds as the first, but with the entire chain of certificates passed along.

In order to limit the usage of the CA-issued certificate and public key of the intermediate delegates, we can decide to not use the CA-signed certificate for authentication by the intermediate delegate for forward authentication, as illustrated with dotted lines in figure 3.5. The identity of intermediary delegate is thus proved to further delegates only with the delegation certificate as signed by the previous entity in the certificate chain.

This also enables the differentiation between traceable and untraceable delegation. In our figure, untraceable delegation is achieved when sending information enclosed in semi-dotted lines. The reference monitor will not have the ability to prove that the information in the intermediary certificates is correct, as the intermediary-identification information is only sent backwards in the chain. If identifying information is not enclosed in the delegation certificates, so that the CSR | IP-A | IP-A for Principal | is replaced with | IP-A | Anonymous for Principal |, the resulting certificate | Principal | Anonymous for Principal | is still a valid delegation certificate, but the end-point will not be able to identify the participants in the chain.

As the identity is always proved backwards, the forward proof of identity in the delegation can be omitted. The trust gained in the forward proof enables the end server to verify the chain of delegation, whereas omitting the forward proof enables anonymity in the chain. Alternatively whether to ask for forward proof or not, can be decided on the fly by the delegate, depending on the chosen implementation.

In our figure, this is illustrated if delegate B only authenticates to delegate C with his | Principal | IP-A for Principal | certificate, and not with his | CA | IP-A | certificate. This means that the only CA-issued certificate in the certificate chain is | CA | Principal |, greatly reducing the exposure of the long term keys of the intermediate delegates, and reducing the certificate chain with one CA-issued certificate for each intermediate delegate.

Traceable delegation is achieved by sending the information enclosed in dotted lines. Then the identity of each principal is always sent forward, and all delegation certificates are issued with CA-signed certificates. In such a scheme, for efficiency, proxy certificates may be preferred in order to save the processing power needed to generate a key. We see that this will work, as in the traceable version | X for Principal | Y | type certificates are never used for signing, they are only authenticated and verified to exist.

An alternative approach is described in [39]. This approach uses an X.509 attribute certificate, a certificate that does not contain a key, only information on the holder of the certificate. Such a certificate may incorporate some or all of the elements of a standard delegation certificate, but without the need of a key generation for each delegation certificate issued. An attribute certificate is used instead of inserting an extension in the X.509 Public Key Certificate. This way the certificate of the delegate can be extended with attributes or information needed for delegation in a dynamic way, without altering the identification certificate of the delegate itself, and with a different lifetime from the identification

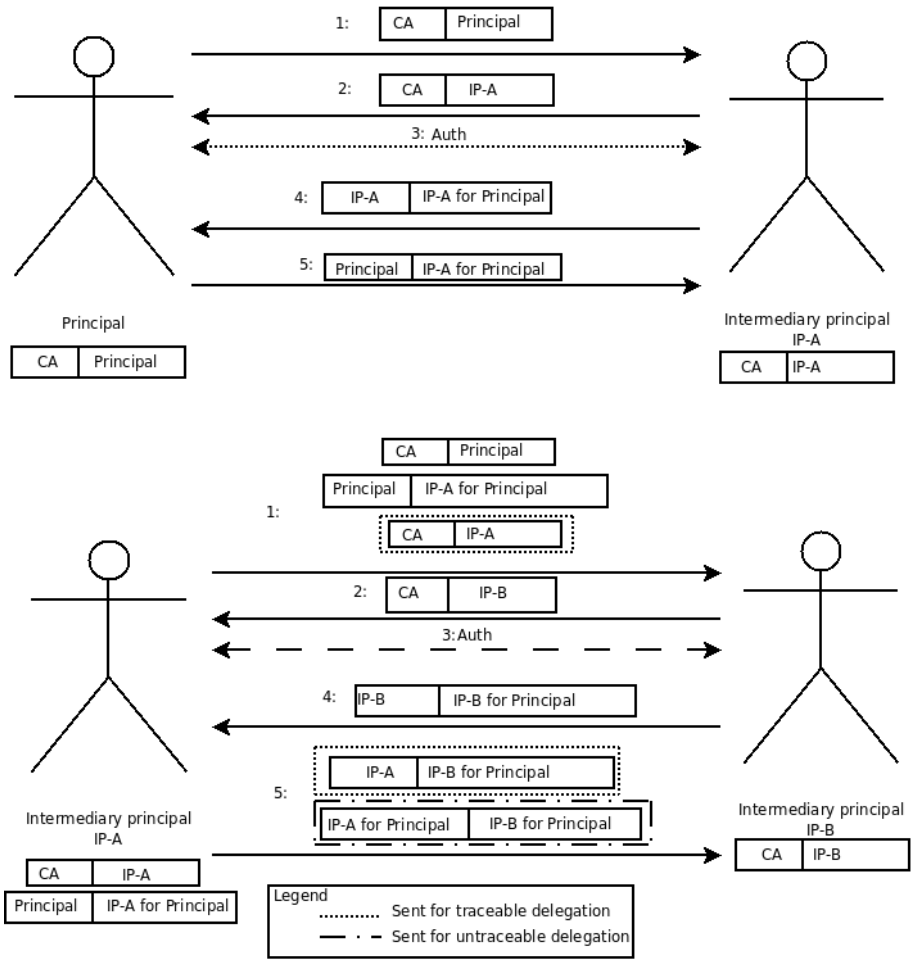

Figure 3.5: Delegation in PKI with proxy certificates

certificate [16]. An attribute certificate is not in itself usable for authentication, because of the lack of a public-private key pair, and needs to be authenticated using a public key certificate.

3.3.6 Advantages of PKI-systems

PKI systems are very good in the basic sense of connecting keys to users. By verifying signatures of preceding certificates in a chain, a certificate can with cryptographic certainty connect the subject with the listed key, and the user can with the same cryptographic certainty prove his possession of the corresponding private key.

In a PKI, a central authority is not needed to confirm identities in the way we see it in Kerberos or NTLM. The user carries with him all the information he needs for authentication, and the only pre-distribution needed is the certificate of the top CA.

A X.509 certificate can also contain a lot more information than just a common name, for instance nothing hinders an organisation from extending the certificate with information about user IDs, group memberships or roles. With such information in the certificate, the reference monitor does not necessarily need to do any network or central mapping of common names to user IDs, and with locally stored objects, the certificate can then in many circumstances be enough to grant access to given objects to principals presenting a valid certificate chain.

3.3.7 Advantages of PKI delegation

The trust model in a PKI is very robust, and is not dependent on any central authority for delegation other than the initial out-of-band distribution of the root certificate. This enables a totally decentralised delegation principle. Moreover, by extending the X.509 certificates as provided for in version three certificates or by adapting the standard to suit another environment, one can embed a number of desired attributes in the certificate, which gives the delegation a high degree of flexibility. Examples of such attributes may be maximum length of delegation chain, and possible end points of the delegation.

At Carnegie Mellon University in the project Grey [6, 5], smartphones are used in a PKI system to enable physical access to offices, enabling delegation of access by signing delegation certificates.

3.3.8 Disadvantages of PKI-systems

In live circumstances, a PKI system can easily get very complex. The issuing CA needs to verify the connection between the subject and the key for each certificate it issues. This process needs to be secure enough to be sure that a foreign key cannot be certified as belonging to a given user.

The verification of certificates is also not necessarily very simple. First of all the verifier needs to have the entire certificate chain in order to be able to do the

verification. This means that the principal may need to transfer a lot more data for a certificate chain than needed in a password system. RFC3280 [31] defines an algorithm for certificate checking that is quite complex, especially if the certificate path includes policy mappings. There is also a matter of certificate validity periods and trust models. As an example of such models, if certificate C_A is issued to be valid between times 1 and 3, and the CA certificate C_{CA} is valid between 1 and 2, should then a signature issued by C_A at 2 but checked at 3 be deemed valid. What if it is checked at time 4? The Shell-, Chain- and Hybrid models, as described in [3] give different answers to this question.

In a password system, a user needs to type in his password in order to authenticate. If a private key is the authentication factor, then the key needs to be stored securely. This can happen on a smart card or by password protecting it on a machine. Nevertheless, if the key is stolen, the entire tree of certificates with the compromised certificate as a root needs to be invalidated. X.509 defines several ways of invalidating certificates, but common for them all is that at point T, to deem if a certificate is invalidated, one needs to check for all certificates in the chain if they are invalidated. Often this is done by checking a revocation list, which requires downloading this list or querying a revocation server. Both these solutions require a certain amount of network interaction.

As a PKI system is secured by cryptographic means, it is also prone to the same cryptographic problems of all public key cryptography. Examples of such problems involves classes of problems where the encryption function is reversible because of errors in parameters, and errors where provoked miscalculations lead to factorisation of the RSA modulus. Further examples of potential public key problems are given in [36, 10].

In case a key in a PKI system is compromised or a certificate otherwise needs to be invalidated, a mechanism must ensure that all validators of the certificate learn that the certificate is revoked. The normal mechanisms for this is to use a certificate revocation list (CRL), described in [31] or an online certificate status protocol (OCSP) request as described in [40].

A CRL is a signed list containing identifying information on certificates revoked, and is issued regularly by a CRL issuer. During the time between the certificate being compromised to the CRL being issued and all the verifiers having downloaded the new list, an invalid certificate may be validated successfully.

The OCSP protocol is an online verification service that is queried for information. The OCSP service is often more frequently updated, but is also a vulnerable point in the system as its presence is required for valid operation.

3.3.9 Disadvantages of PKI delegation

The paper by Welch et.al. [52] details delegation in X.509 by giving each delegation certificate its own separate key to allow a less secure treatment of the delegation private key. The creation of a public-private key pair is resource demanding, and in a large data center the generation of keys may be one of the most time consuming process of delegation. Public key algorithms are also always more resource demanding than their symmetric-key counterparts in both key generation and key usage.

Issuing an identity certificate requires verification of the identity of the certificate holder. In a large data center, such verification may be hard to automate without manual intervention.

In Kerberos, each access token is readable only by the holder of the secret key, and the knowledge of the session key is thus dependent on being able to decrypt the token. In a PKI environment, presenting a certificate is not proof of identity, and a PKI delegation scenario thus always needs mutual authentication between the service and the principal.

All certificates in the delegation chain needs to be sent to all hosts that wants to verify the delegated credentials. This means that the length of the authentication increases proportionally to the length of the certificate chain. The measured length of the certificate shown in listing 3.1 is 1093 bytes in its binary form. As the certificate does not contain an especially long key and not an overwhelming amount of information, we see that a certificate chain will most likely grow by a factor of at least 1093 bytes for every certificate in the chain, and the authentication process will require more data transmission and processing compared to the relatively small data exchange of Kerberos.

3.4 Grid systems

In later years collaboration internationally, especially in academia, has led to a lot of research in the field of grid computing [19]. A computing grid is in many ways different from a data center, but access control systems for grids carries a lot of properties interesting for a large scale access control system for a data center. Amongst these properties is the fundamental design criteria that the grid system is a large scale computer system with diverse hardware, principals and objects from different trust domains and different physical organisations, and dynamic establishment of trust relationships.

Seeing that these properties of the system also go beyond the basic authentication in a data center, it is important to include this section for the sake of comparison. We will see that the basic solution for authentication is a PKI solution, and therefore that the analysis of PKI covers the role of grid computing for authentication in a data center.

The Globus Alliance [All08] points to a definition of the grid concept as it is stated in the paper by Foster et.al.: «The real and specific problem that underlies the Grid concept is *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*» [22], and the Globus Alliance makes a software toolkit for «fundamental technologies behind the "Grid," which lets people share computing power, databases, instruments, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy.»

The core of grid computing is the «virtual organization», a collection of collaborators and resources physically present at different locations and in different computer systems in a virtual computer system. The virtual organisation aims to enable access to both systems and data resources, and to enable collaboration between different organisations participating in the virtual organisation. This

collaboration of computer systems needs to enable access control in a distributed environment without central administration, and collaborate with diverse access control systems in several computer organisations without being part of the organisations itself.

A grid needs to be able to create services, access-groups and trust-domains dynamically, while still enabling accountability for resources used, resource quotas, the physical organisation the users belongs to and other properties the parties in the virtual organisation deems important to a productive and fair cooperation [53].

A prominent goal of all grid computing services is to be able to cooperate with the user's local access control systems.

3.4.1 Grid Security

The biggest challenge in grid security is to be able to support a large scale dynamic and diverse computing environment while adhering to the participating organisations' security policies and rules. According to Welch. et.al. the key attribute of grid security is to enable access to resources and objects for grid users governed by the participating organisations local security policies, policies which talk only about local users. Thus the objects and resources need to be accessed through trust existing between the local users and the objects, and between the local users and the virtual organisation [53].

This is done by creating the grid as a 'policy domain overlay', whereby the institution outsources certain aspects of its control to the virtual organisation, who in its turn unifies all outsourced policies from all members in the virtual organisations in one policy, allowing the network to behave in a homogeneous fashion to all users.

Welch et.al. [53] together with Foster et.al [21] further describe three key functions in a security model for grid computing:

1. Grid security needs to interact and be interoperable with the diverse local security mechanisms in the participating physical organisations and laws in the local countries of the participants.

2. The user population and the resource population is dynamic and often large and spread over a large geographical area.

3. Grid security needs to handle dynamic creation of services by individuals in different physical organisations and enable security in these services in the grid without undermining the local policies.

4. Grid security needs to enable dynamic creation of trust relationships within the grid, both between users and services, but also between services themselves, so that the grid may be efficiently coordinated. The trust domains need to be able to adapt to users and organisations joining and leaving the grid.

Grid computing require a user-driven security model to enable such dynamic security mechanisms [53]. This means that users of the grid must be able to set up new services and administer access rights on them without central administration privileges.

3.4.2 Implementations of grid security mechanisms

The Globus Toolkit version 2 uses the Grid Security Infrastructure (GSI), which provides a security model based on X.509 certificates and TLS. The diversity in local security mechanisms is solved by using a security mechanism gateway, a gateway capable of translating between different security credentials, for example translating from X.509 to Kerberos. This way a local service governed under Kerberos may be accessed by a grid user using a X.509 certificate by using the security mechanism gateway to translate the X.509 certificate into an appropriate Kerberos ticket.

Globus Toolkit version 3 utilises the open grid services architecture (OGSA [20]), technology to implement grid security on the basic foundations of web services. Many of the features also found in GSI, amongst them the access token translator, in OGSA called the credential conversion service, are continued as services. OGSA also adds a number of other services to the security infrastructure. Mechanisms for services to publish its security policy and to allow service discovery and a service for authorisation are amongst them, offloading from the service itself the need to authorise clients. The authorisation service works by examining the authentication credential presented to a service and uses knowledge of the policies of the grid and the local organisation to allow or disallow access. All these services are run in all local organisations participating in the grid, and use X.509 certificates as a trust basis for grid services over organisation boundaries.

An overview of a connection in Globus Toolkit version 3 is given in figure 3.6, as taken from [53]. First the client's hosting environment checks the published policy of the requested service at step 1. The correct credential type is requested from the credential conversion service in 2 and checked and handled by the token processing service, 3. The requested service then also uses a token processing service to handle the token in 4, offloading all token handling from the applications to the token processing service. Finally the requested service uses the Authorisation service to make policy decisions in 5. Detailed explanations of Globus Toolkit version 3 security and OGSA are found in [53, 23].

3.4.3 Delegation techniques in grid computing

The core of grid computing delegation, as detailed in [21], is the user proxy. A user proxy is a machine or process responsible for representing the user if the user is not available. The user proxy is given a delegation token by the user, $C_{UP} = \langle user - id, host, start - time, end - time, auth - info, \ldots \rangle_U$. Together with its own credentials, the user proxy can then use this token to authenticate as the user in the time-period detailed in the token itself.

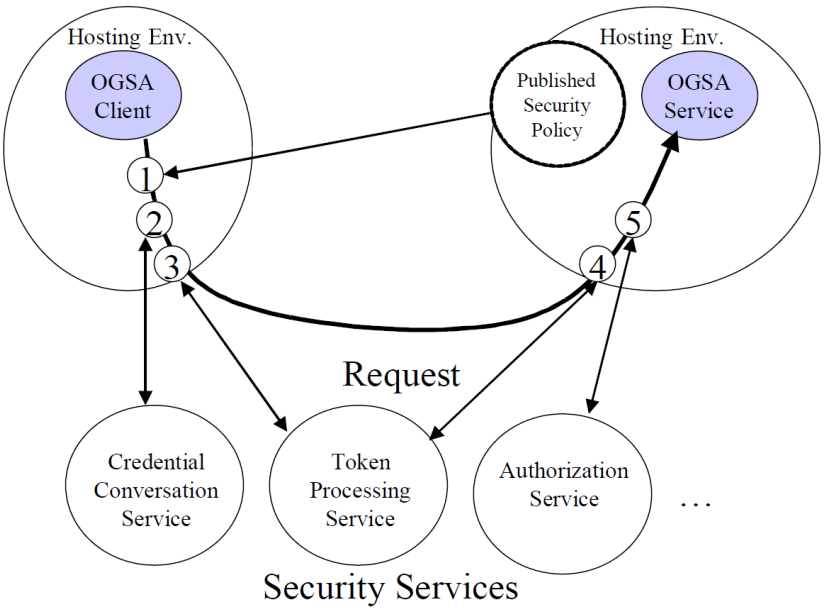

Figure 3.6: Overview of connection steps in OGSA, from [53]

The concept of delegation in the grid is equal to one-step delegation in PKI with attribute certificates, and shares the same advantages and disadvantages as delegation in PKI.

Chapter 4

Analysis of fault handling and administration aspects

In this chapter we will look at administrative aspects of Kerberos and PKI. We will discuss the trust in the system and propose solutions that may solve some of the administrative problems of large scale access control systems. We will also analyse fault handling aspects of Kerberos and PKI. Finally, this chapter will provide necessary background information for the performance analysis in chapter 5.

The features to be analysed in this chapter are:

- Procedures for enrolling new users and new machines in the data center
- Services that span multiple machines, and how they can be set up dynamically
- Revocation of entities in case of compromised keys
- Failure handling

4.1 Trust

All access control systems require some elements to be trusted. Informally, an element is trusted if it works according to the security intentions of the element owner. For example, a trusted service would be a bug-free, security-flaw-free, and uncompromised service, and a trusted channel between two elements would reliably transfer messages between the two elements, and only those two elements. Focusing on authentication systems, trust may be required in a central login system to tell the real identity of a logged on user, or to deny login to unknown users or users who presents false passwords.

When trust in an object depends on trust in a different object, we call it a chain of trust. An example of a chain of trust can be given in a local access

control system. A user expects the local access control system to give file access only to authorised users. Thus, if user 'alice' has a file called 'secrets' and has allowed only user 'bob' and herself to see it, she knows that as long as the system is trusted, the file is not accessible to the user 'eve'. The access control system depends on trusting the login component. If the login component's trust is broken, and user 'eve' logs in but the login system tells the access control system that the user logged in is 'bob', the file 'secrets' becomes readable to user 'eve', not because of a fault in the access control system but because the access control system depends on trust in the login system, which is broken.

4.1.1 Trusted service

In both Kerberos and PKI there are central services that need to be trusted. In Kerberos the KDC needs to be trusted to provide correct authentication tickets. In PKI the CA needs to be trusted to only issue certificates that bind the correct identity to the correct public key. If any of these central systems are compromised, the system as a whole cannot be trusted, as the entire system depends on the trust in these central components.

It is vital for the security of the system that all machines and services are correctly identified in the system. It is also important that when a credential is given to a system, the credential must not be readable or forgeable by any other system in the data center. Each machine also needs a service accessible to system administrators to enable administrative tasks to be performed. In addition, in a data center, each machine may be running different services at the same time. We need to make sure that the secret key of all services stays secure within the machine they are running on and is not stolen by other services running on the same machine. We also need to make sure that the same protection is established for the session keys and on any forwarded tickets from the clients. This implies the need for a trusted point of contact on each machine.

We propose that each machine is fitted with a trusted service that is responsible for handling the secret key of the machine, and also functions as a remote administration service enabling logon to the computer to perform administration tasks such as updating software and setting up new service instances.

A trusted service enables us to model a breach of machine security. We model the trusted service as having complete control over the machine, even though the implementation of the trusted service may not directly link the exploitation of this service to achieving complete control of the machine. We still model the trusted service to have this capability, as at least one program, the operating system, must in all cases have complete control over the machine.

4.1.2 Machines that run many services at once

The trusted service can safely be modelled to be a part of the machine itself, as we model the trusted service to be the point of contact into the machine for administration tasks such as distributing updates or setting up new service instances, thus linking the trusted system closely to the operation of the machine. We therefore use the word trusted service and machine interchangeably in all

vulnerability aspects of the data center, because in this model, compromising a machine is the same as compromising the trusted service and vice versa.

4.1.3 Consequences of trusted service compromise

If the trusted service of the machine is compromised, no information sent to or coming from the machine can be considered valid. All encryption keys the machine knows must also be considered compromised. In Kerberos this means that if a machine running a file service is compromised, the file service session will be compromised. If no additional actions are taken, the attacker will have access to all data the in-session user has access to, for as long as the delegation ticket is valid. In addition, if the file server was delegated to, the attacker may also be able to further authenticate as the user. If data is stored locally, the attacker may read all this data, and if there is a trust relationship between the file-service instance and other file-storage nodes, the attacker may exploit this relationship as well.

The trust in the trusted service does not need to be absolute for the security of the data center as a whole, as the trust in a computer is not transferred in the administrative system of the Kerberos or PKI system. If a trusted service is compromised, the trusted services on other machines are unaffected.

Even though the trusted service establishes an additional attack point in the system, any system running administration services will have the theoretical possibility of losing complete control to an attacker if a sufficient weakness exists in any of the services on the machine. We also assume that the data center can not function without the possibility of remote administration of the machines via an administration service. We see that this chain of trust via a trusted service on each machine does not weaken the theoretical security of the system, as a compromised operating system in all cases would yield all active sessions and all local keys to the attacker.

The consequences of breach of trust in the trusted service means that a non-trusted computer, such as a computer inserted into the data center by an untrusted source such as a customer, must not be delegated to by any user without the user acknowledging the consequences of lack of trust in the machine. Such a machine must also not run services that implies confidential data to be uploaded to it.

4.2 Initial trust establishment for new machines

In a data center, new machines will be introduced. In a running state this will often be because of replacement of failed machines, but it may also be due to other reasons, e.g. the data center expansion. All these new machines need to establish trust relationships with the central service in form of having a certificate issued by the CA in PKI or establishing a shared secret with the KDC in Kerberos. Initial trust establishment is a problem in all distributed trust systems, and in this section we propose a method that may be used to establish first-time initial trust in a data center.

In both Kerberos and PKI each machine in the system needs to have its own key to authenticate the machine and the administration service. With our trusted system, the key for the machine and the key for the administration service are the same. We propose that a central installation image exists that gives all computers the identical initial software, and seek a solution to enable each machine to acquire a unique private key that is authenticated by the CA for the PKI scenario and a shared key with the KDC in the Kerberos scenario. We want this solution to be as automated as possible to enable remote initialisation of new computers, and thereby limiting the job of the technician to installing the initial software image and plugging the computer into the network.

There are several possibilities of how to establish this initial key. The most basic way is using manual means of key distribution, where the initial key is distributed via a CD or USB-stick by an operator. In a large data center this can easily be too labour-intensive, and errors may easily occur in cases where media is broken or misplaced or exchanged by human errors or otherwise.

If automation is required, Diffie-Hellman (DH) key exchange [13] has the advantage of automatic key setup. This is also true for a standard CSR. CSRs and DH may be utilised either with or without additional authentication. DH without authentication is a method highly vulnerable to man-in-the-middle attacks, but as this is a one-time procedure, and in a company-controlled network, the risk may be determined to be acceptable under certain assumptions about the likelihood of being able to perform such an attack. For CSRs, as they are authenticated by a secret key, the integrity of the certificate is protected during the transfer of the CSRs. However, as the identity of the machine is embedded in the certificate, anyone can generate a CSR and embed a suitable identity. This means that the CSR approach needs to have additional checks for the identity of the subject.

We propose that an authenticated and integrity-protected DH key exchange or CSR authentication may be performed by using a serial number from the computer hardware as an integrity key in in the process. In an example scheme, a hash of the serial number will identify the new machine to the TGS, and the serial number may be used as the integrity key. The TGS will be informed of the new computer and the expected serial number, and will use the hashed serial number to know which computer is signing on.

This procedure is outlined in algorithm 4 both for Kerberos, using DH to agree on a Kerberos key and for PKI by authenticating the transfer of the CSR. Alternatively a PAKE-scheme (password-authenticated key exchange [7]) may be used instead of DH. As PAKE is resistant against dictionary attacks, potential problems of using serial numbers drawn from a small pool are reduced.

By not directly using the serial number as a shared key, hostile services on the machine cannot use potential access to this information as a source of key information, and as a serial number is not machine-accessible from anywhere else at the time of computer installation, this information can be used as authentication. An attacker needs to know a correct serial number on a piece of hardware in order to successfully complete a man-in-the-middle attack on the DH algorithm.

Algorithm 4 Authenticated key exchange using hardware serial numbers

Kerberos and PKI:

1. One hash function is chosen in the data center for initial authentication purposes, H, and one authentication scheme is chosen, $A(k)$, using a shared key k.
2. When the machine supplier delivers machines, an electronic list containing serial numbers of pre-determined components such as CPUs or USB controllers are also delivered to the central administration system (CAS).
3. The data center technician installs the software image on the computers and plugs them into the network
4. The trusted service from the machine reads out the serial number S from the hardware components
5. The machine reports to the CAS, identifying itself with $H(S)$

For Kerberos:

6. The CAS sends out authenticated DH parameters, $\langle a, g \rangle_{A(S)}$
7. CAS chooses a secret, e, and the machine chooses a secret, d
8. Both parties chooses a nonce each, n_1, n_2
9. $\langle n_1, a^d mod\, g \rangle_{A(S)}$ and $\langle n_2, a^e mod\, g \rangle_{A(S)}$ are exchanged
10. Both parties authenticate the messages, and check that they did not receive their own message in return.
11. Both parties agree on the key $a^{ed} mod\, g$ using the knowledge of their own secret value

For PKI:

6. The machine generates a CSR
7. The machine sends $\langle CSR \rangle_{A(S)}$ to the CAS
8. The CAS gets the CA to issue the certificate based on the CSR
9. The CAS sends the certificate back to the machine
10. The machine verifies the signature on its issued certificate

4.3 Initial procedures for new users

In a data center there will also be users that come and go. All users need to set a password in the data center or establish a credential in some way, for example by getting a shared secret key in a USB stick or having a certificate issued on a smart card.

The process of inserting a new user in the Kerberos data center involves the user setting his password in the AS. This enables the AS to issue a TGT encrypted with the key derived from the password. This process needs to involve a trusted path in order to make sure that the user's password is not compromised on its way to the AS. The procedure may consist of a transfer of the password to the AS via a trusted path from another system, such as a central user database, or direct entering of the password via a trusted service on a dedicated system. The passwords in the system may also be centrally generated and sent to the user on a rub-off card (a card where an opaque film needs to be scratched off to reveal the text) so the user can see that the password is not compromised in transit. The setting of an initial user password is not conceptually different in Kerberos than in any other central access control system.

Issuing certificates to new users in PKI is different to the Kerberos solution where each user authenticates with a password. In PKI each user needs to hold a signed public key and the corresponding secret key. This necessitates the use of a data storage unit for each user, often implemented by using a smart card with a key encrypted by a PIN, but it may also be implemented using a key-storage system on the users computer.

If the smart-card solution is chosen, smart cards may be issued to the user without user input. The user receives a smart card, and is able to use it as soon as the PIN is set by the user or otherwise conveyed to the user. This method of user authentication by centrally issued smart cards is used amongst others by Technische Universität Darmstadt for their PKI [15] and in the GSM standard [45, 51]. Having both a card and a PIN also makes this a two-factor authentication system, strengthening security assumptions over a single password system like Kerberos.

If the local key-storage system is used, a smart-card may still be issued and the key transferred to the user's computer. Alternative solutions involve transfer of the initial certificate and secret key to the users computer via physical media such as CD-roms or via the trusted service. The secret key may in this process be encrypted by a password given to the user via out-of-band means, such as on paper.

4.4 Service instances spanning several machines

In a data center there will be services that are present on several machines for load balancing, and where the client does not know beforehand which server to contact. Examples of such services may be database servers or file servers. The client may also need to contact such server on different machines during the same job.

In this scenario in a Kerberos environment, if a ticket is issued for fileserver@machine1, and the load balancing algorithm orders the next contact to fileserver@machine2, the previously issued ticket for fileserver@machine1 will not work to authenticate the user on fileserver@machine2 if the service instances have different secret keys.

There are several possible solutions to this problem. One solution is to require the issuing of one ticket for each machine the file server runs on. This means that the client needs to be given a final instance address to the desired service before contact is made. This can be done by using an indirection service directing the client to a given machine before a ticket is asked for, or embedding this service in the Kerberos system itself and embedding the name of the correct machine in the Kerberos ticket. In both these scenarios, access to the service may require frequent reissuing of tickets, depending on whether or not access can be done to only one service instance during the entire ticket lifetime.

A different solution is to share the same Kerberos secret on different machines. This means that a ticket for 'file server' is valid to any instance of the file server, regardless of which machine the instance is running on.

Both these solutions have their strengths and weaknesses. Certainly having one secret per machine is the more secure way. This enables the revocation of the secret in case of compromise of the trusted service or the service instance running on the machine, and it prevents the other instances of the servers to read the contents of the messages passed to the active instance by the client, and thus hinders a compromised instance in eavesdropping on any active service connection. The disadvantage of this solution is that it is dependent on a visible load balancing system. The client needs to know directly on which machine to access a service, either by way of getting this information from a load balancing or addressing service in order to be able to request a specific ticket, or by embedding this information into the Kerberos server and passing the machine to contact in the message back to the client together with the ticket for that specific machine. This avoids the need for an extra connection to the indirection layer.

The shared secret solution enables the possibility of a transparent load balancing system and the seamless dynamic deployment of instance resources. If the service has a shared secret, a system can be made that spawns extra instances of services on extra machines in times of high load, for example, and without the need of setup-overhead with the Kerberos system. The possibility of load-balancing can also be enabled without the need to contact the Kerberos server for each redirection the connection to a service instance will go through.

A single shared secret will be highly vulnerable, and the probability of compromisation of the secret will increase with every machine the service runs on. If the service is deployed on one single compromised machine or the key is broken because of one unfortunate event, the entire file server system is compromised. For the rest of the thesis we will assume that each service has its own separate Kerberos key, but the load balancing system that enables this is outside the scope of this thesis. This means that the service will always ask the KDC for a specific file service instance.

PKI does not suffer from the same issues in multi-presence services as Kerberos, as service certificates are always used to identify any service, and at the same

time provide the service key included in the certificate. With a service running on several machines, contacting the server instance on any machine will result in the reception of a public key and a certificate that can be authenticated to identify the owner. Thus contacting any new instance will result in the reception of the verifiably correct key directly. This way the same key does not need to be shared between multiple instances of a service.

As long as the owner of a certificate is deemed to be the service that was contacted, the transaction will succeed, regardless of the key in the certificate. As an example of this, if a principal needs to contact a file service it will receive the file service instance certificate from the chosen node. The certificate will indicate that it is a file service, possibly by being issued by a file-service master to be discussed in section 4.5, and the public key of the instance is embedded in the same certificate. If the certificate is verified to be correct, the public key can be used for the session. As this certificate is received and verified for each service contacted, whether the key is the same for all services does not matter, and we can choose them to be different.

If caching of certificates are used, each certificate may be given its own serial number, even if the common name in the certificate is the same for all instances of the file server. This will identify the key that the currently contacted server intends to use for communications.

The life time of the instance certificate depends on the revocation method chosen, but may be small enough to protect the system if the revocation system chosen is to not do revoking at all. The instance will renew its certificate with the master before it runs out. If any master learns that an instance certificate is compromised, the instance will not be issued a new certificate.

4.5 Procedures for new services and service instances

All data centers need services running on its machines. These services may be anything from file services to calculation services, and will run on any number of machines. We here propose a possible method for introducing to a Kerberos system a new service or a new instance of an already existing service on a new machine. We also look briefly at how to add services in a PKI network, but this problem has been thoroughly examined by others [All08]. Unlike the Kerberos implementation from MIT [MIT08], which uses administrative commands on both the KDC and the designated machine to register new services, our method enables dynamic registration of new services by any user in the Kerberos system.

Introducing a new service in the data center includes agreement of a key with the Kerberos server or issuing of a certificate to the service instance by the CA, and in both cases it also includes identification of the service in the data center. In Kerberos, for identification and key distribution we propose a solution involving the registering of a service in the Kerberos system by a user or a service-owning master service. A service is given a verifiably unique name in the data center. The new service instance is given its own entry in the Kerberos database under

its unique name, and the rights under which the service runs are therefore independent of the owning user.

The method relies on administrative commands to the Kerberos server not found in the Kerberos protocol as it is described in [32], but in a separate administration protocol. Normally such a process is also dependent on access restrictions, but the authorisation processes in this procedure are outside the scope of this thesis.

To initiate a service instance on a server, the user or master service first authenticates to the KDC and reports that it wants to start a new service under a given name A. In case the name A is already in the Kerberos network, the KDC denies the request. Otherwise the request is granted. The KDC then gives the user or master service the unique key for the service instance.

Further the user or master service authenticates with the machine they want the service running on. They upload the service executable together with the service name and the service key. Upon starting, the service uses its name and key to register with the KDC in order to let the KDC know of the location of the service.

An example of this is again the file server. If a new instance of a file server is needed, the file server master service authenticates with the KDC and reports that it wants to start a new file server instance under the name 'file1'. The KDC issues the master service a key for the new instance. The master file service will then proceed to upload the executable and the key to a machine, where the service will register with the KDC.

In the PKI scenario, each initiating instance owner will authenticate to the trusted service on the machine he wants the service to be running on, and proceed to upload the service instance program. Upon startup the program will generate a CSR and send it back to the instance owner. The instance owner will issue a certificate based on the CSR with his own certificate. With this procedure, any principal wanting to contact a file service instance will check that the certificate of the instance is issued by the correct master service. This is in essence the same approach as taken by the Globus alliance [All08]. It is outside of the scope of this thesis to discuss methods for finding instances of a given service in a PKI environment.

4.6 Routines for certificate revocation

If machines are compromised in the data center, keys will be considered compromised. Key compromise can also happen for other reasons, but regardless of the reason for compromise, a compromised key will enable false authentication in the data center under the credentials of the user whose key is compromised, or loss of confidentiality of data that has been transferred and encrypted with the compromised key. The recall of a key is called a revocation, and may also have other reasons than key compromise, for example loss of the private key.

For Kerberos with a central key service, invalidating revoked keys is a matter of deleting them from the central database, or alternatively to flag a ticket as

invalid. For PKI there are several techniques to let a community know about key revocations, and we here discuss key revocation solutions for a data center.

The X.509 standard has several mechanisms for certificate revocation notification, as mentioned in section 3.3.8. There has been much research in the field of certificate revocation [58, 37, 38], and a partial overview of different methods is given below.

A CRL contains the revocation data for certificates and a reason for the revocation, in addition to other elements like the date of issue of the CRL [31]. By checking a CRL the verifier of a certificate is able to see if a certificate has been revoked. A different method of revocation checking is to query an OCSP server. This query will result in a signed answer from the OCSP server containing the revocation status of the queried certificates.

As every proxy delegation certificate is issued with the goal at hand to be short lived, checking the proxy certificates may be omitted in situations where the validity of the certificates are older than a given threshold. Assuming that long jobs in the data center are quite static, this will not affect the startup time of large jobs, nor have a huge impact on the OCSP-server in cases of re-logins late in the process.

The cost of checking a CRL lies with each client and on the network. As every client in the data center needs to have a copy of the entire CRL, downloading this CRL may have a big impact on the data center, depending on choices for delta-CRLs and update-times. The time from revocation until all clients have acquired the updated CRL is also an issue in this scenario, as it is presumed that if a key is compromised, an attacker will try to use the key as quickly as possible, before the CRL can be propagated in the network.

A third option for revocation is described by Hayton et.al. in [29], where each delegator maintains a delegation record, pushing events down the delegation chain as they occur. In the paper this is implemented in a role based access control scheme, where each role may depend upon several conditions. The verifier of a certificate will ask all the authorisation authorities for the set of conditions to verify that the conditions hold, and to push changes in conditions if they change. For example, if a client needs to have conditions A and B met to be able to achieve access service C, the issuer of A and B will be queried by C for the validity of the certificates for A and B, and the issuers will also later inform service C if the validity of A or B changes. A tick-service is also implemented, such that a presence-notification from the issuer of A and B will be sent at regular intervals, enabling C to know that the notification chain is alive, and take appropriate action if the notification service falls out and the conditions of A and B becomes unknown.

This form of cascaded revocation seems not to be suited for a delegation chain environment. As only the latest principal is authenticated by the service, and the previous links in the chain are authenticated by the validity of all signatures on the last certificate in the chain, the system will be exposed to problems if a system fails and a temporary key is lost, for example because of a computer failure, or if a system needs to be relocated or used for other purposes. As a job started by a delegate does not need to return results back to the starting delegate, but may for example update a database with its results, scenarios

may exist where the intermediate delegates may not be part of the running job anymore. A machine that is not part of a job can not be a part of the state of the job.

An alternative approach to revocation is to not do revocation. If all certificates have short validity times, certificates may be replaced frequently. The replacement may be carried out along the procedures of initial issuing, but authenticated with the expiring certificate. This approach, while sacrificing security, saves the data center from all costs of revocation checking. In addition, since all revocation checking is delayed by the process of doing the actual revocation, with short certificate life times the period of insecurity from key compromise to revocation may not be significantly shorter than the time span from key compromise to certificate expiration. Moreover, so long as the key is secure, new certificates may be issued with the same key, eliminating the cost of key generation.

As with the CRL method, if a certificate is compromised it needs to be replaced. This replacement is not conceptually different whether a CRL exists or not. A replacement can follow the procedures for initial issue, but may need manual authentication in case the serial number or other authentication means is compromised along with the certificate.

4.7 Failure handling

All data centers will experience computer failures, and all access control systems need routines for handling situations that occur when parts of the data center fails. This section will describe which features exist in Kerberos and PKI to enable graceful failure handling.

4.7.1 Failure handling in Kerberos

Reliability and security of the Kerberos system can be based on using distributed servers and failover servers. As discussed in section 3.2.2, this can easily be done as the Kerberos protocol stores all session state in the tickets. This is also the reason why Kerberos will scale roughly proportionally with the number of Kerberos servers available, provided that a load balancing system is used to distribute the load between them. As Kerberos is designed for scaling to indefinitely many nodes [41], the correctness of this calculation is strengthened.

Replication of Kerberos servers carries in it the inherent problem of distributed synchronous databases. If the Kerberos servers are truly independent, this means that the key and access information databases needs to be synchronised between them, to make sure that all entities in the system can utilise all Kerberos servers at any time.

Another negative with fault tolerance in Kerberos is the fact that all access requests need to be handled by a central system that may fail. If the servers are overloaded or the load balancing algorithms do not work, the Kerberos system will have bottlenecks that cannot be overcome without adding additional resources in the data center.

As well as creating bottlenecks, the reliance on central servers in a very large scale data centers has consequences in terms of stability of the system. Provided that the Kerberos system is composed of several instances of the same server, failures in the system may take the whole data center down more frequently than calculated as failures in homogeneous systems are not independent [11]. For example, a certain Kerberos implementation may make all systems fail on February 29, leading to a complete data center failure. On the other hand, having heterogeneous components increases the complexity of the system. It will be important to rigorously test any implementation of Kerberos servers for large data centers to avoid failures of too many Kerberos instances at once.

4.7.2 Failure handling in PKI

As PKI does not depend on the current liveness of a central server, there is no obvious bottleneck in the system. Failures in PKI will hence only affect the node that fails. One possible bottleneck in the system, if implemented, is the CRL server or the OCSP server. The OCSP server will have to verify an astounding number of certificates if used, and the CRL server will have to upload a huge amount of data to the network every time the CRL is updated. The possibility of push-based revocation cannot be considered scalable as this may keep machines that are no longer part of a running job a component of the job state.

Chapter 5

Performance analysis

In this chapter we will analyse the performance of some of the described technologies in a large scale scenario. The two technologies compared are the two main technologies seen in use today, namely Kerberos and PKI. Kerberos is implemented in many computer systems, amongst them Microsoft Active Directory, and PKI is found also in large scale grid systems.

It is not a goal in this analysis to evaluate an existing implementation of either Kerberos or PKI. This is because a large data center in a company in all cases would have to adapt and implement one of the discussed technologies to its own needs, not at least because of the size of the data center. This thesis will therefore focus on the abstract properties of the chosen technologies, and not focus on protocol details (of the technologies). This is also because functionality in the protocols can be used in different ways to achieve the same goals, for example to enable delegation.

5.1 Performance analysis scenario

In order to analyse the performance of the different technologies in a multi-thousand machine data center, we will make the following assumptions about the data center environment. The number of computers is based on estimated numbers of computers in Google's data centers from Baseline Magazine [Car08] and a job posting from Microsoft announcing an open position as software engineer for the «Autopilot» program, a data center application they announce to scale to 100 000 computers [Mic08a].

- The data center contains 100 000 machines
- There are 1000 users
- Each user runs up to 10 jobs per day.
- Each job runs for an extended period of time, on an average of 1000 machines, and a maximum of 10 000 machines.

- Jobs may be dynamic in size, e.g. a job may start out on 10 computers but grow to 1000 without interaction with the user's computer
- The jobs may access services in the data center on the user's behalf (using delegation), such as file servers
- Each machine in a job accesses on average 10 other services during the execution of the job.
- Computers in the data center will fail, and the jobs may need to be relocated and restarted without interaction with the user's computer.

We will analyse the technologies chosen with regards to the following metrics:

- Storage requirements
- Computational requirements for the average load and for the peak load of a maximum sized job of 10 000 machines all accessing a single service at once
- Communication requirements

5.2 Performance of Kerberos

In this chapter the performance of a Kerberos system under the written assumptions is analysed.

5.2.1 Storage in the KDC

The access information database on the KDC in a Kerberos system is dependent on several factors. The number of users, the number of machines and services in the system and the chosen size of the Kerberos key. The Kerberos server may also need to hold some access right information associated with each user, such as ticket issuing policies. Even though this access information is outside the scope of this analysis, the total amount of data stored for each user is important in the analysis.

Each entity involved in Kerberos needs its own shared secret with the server. Our system has 1000 users, which results in 1000 user keys. Furthermore, the server has at least 100 000 machine keys, one for each machine in the data center. Our KDC therefore needs to hold minimum 101 000 keys. Service instance keys comes in addition to this. If we assume an upper bound of 4 000 000 service instances in total in the data center (i.e., 40 service instances per machine, which is probably a gross overestimation), the KDC would need to hold 4 101 000 keys.

Estimating the size of the information held for each user, we note the Microsoft knowledge base article [Mic08b] on solving the problem of sending Kerberos packets larger than a UDP packet of 1400 bytes. The article implies that in some cases the Kerberos messages are bigger than 1,4KB, but that they normally are smaller. We incorporate the uncertainty of packet, key and access block sizes

by estimating the size of tickets, packets and user right information size all to 1KB.

The upper bound on the size of the information database is calculated by setting each entity's access entry in the database to 1KB and calculating with the maximum number of 4 000 000 server instances in the data center, giving 4 101 000KB, or 3,9 GB. This is less space than available on a single RAM chip available in any computer store.

5.2.2 Resources spent on ticket issuing

A maximum bound for the number of tickets issued per day can be found by assuming that all users start the maximum number of jobs per day, that the jobs are at maximum size and that all machines in all jobs access 10 services each. In addition, all machines need to be delegated to in order to be able to utilise the services. This means that each machine in all the jobs needs an extra ticket. This gives a maximum number of tickets issued per day of 1 100 000 000.

$$1000 users \cdot 10 \frac{jobs}{user \cdot day} \cdot 10\,000 \frac{machines}{job} \cdot \left(10 \frac{services}{machine} \cdot 1 \frac{ticket}{service} + 1 \frac{delegation}{machine} \cdot 1 \frac{ticket}{delegation} \right) = 1\,100\,000\,000 \frac{tickets}{day}$$

For the estimates on the average load of the Kerberos server, we use the average number of machines per job, 1000, equalling 110 000 000 lookups per day, or averaged to 1270 lookups per second.

For each utilisation of a service in the data center, the TGS needs to generate one random number for the session key, do one lookup of a key to encrypt the ticket for the service, do the encryption of the ticket and do the encryption of the packet. We assume that the two encryptions are the dominant process in this algorithm, and use the timing measurements form [Dai08], where AES is measured to encrypt 84MB per second in CBC mode with a 128 bit key, in addition to using 0.45µs for the setup of key and initial vectors (IV) for each encryption.

The time spent each second on key processing, neglecting the random number generation and RAM lookup time, can be estimated by the number of encryption setups, setting the amount of data to encrypt to 1KB. The 1KB data amount also incorporates the variance in size of the ticket and the total server response including the ticket. The total time for handling 1270 hits is about 31,3ms as seen in table 5.1. From the table we also see that when all 10 000 machines in a single job try to access the same service at once as our defined peak load states, requiring the KDC to issue 10 000 tickets, the KDC can easily handle this within one second. We note that if the peak load is defined to be all 10 000 machine trying to access all their services (average 10 per machine in our scenario) at once, a single Kerberos server would not be able to handle the resulting load within one second.

Setup time for one encryption	$0,431\mu s$
Amount of data to encrypt	$1KB$
Encrypt 1MB with AES/CBC 128	$\frac{84MB}{s} = \frac{0{,}01162s}{MB}$
Total time pr. ticket	$\left(\frac{0{,}01162s}{MB} \cdot 1KB + 0,431\mu s\right) \cdot 2 \approx 24,11\mu s$
Total time for 1270 tickets	$30,62ms$
Total time for 10 000 tickets	$241,13ms$
Max possible tickets pr. second	$\frac{1s}{24,11\mu s} = 41\,476$

Table 5.1: Time spent by the TGS to issue Kerberos tickets

For Kerberos any modern symmetric key standard is applicable. As seen in table 2.1 and appendix A, all symmetric key schemes are quick enough to handle basic loads, and all AES key lengths are quick enough to handle even the load created by 41 476KB of data per second. As encryption of 84MB/s under AES/CBC 128 is slower than 10GBPS IEEE 802.3an-type ethernet [1] presumably found in any large scale data center, the amount of data to send out on the network is limited to the speed of the cryptographic operations in this setup.

5.2.3 Performance in delegation scenarios

As described in section 2.2, processes expand in a tree structure. For each expansion in the delegation tree, the intermediate principal needs to get at least one ticket for authenticating further. If the next step is another intermediate principal, a new forwarded TGT needs to be issued. If the new intermediate principal should be able to authenticate further but not delegate further, a proxied TGT needs to be issued. If the next step is the end point, no extra tickets are needed. If by default a delegation is made on the basis of not knowing beforehand whether the intermediate principal needs to access further data, and to allow for the worst case scenario, each machine in use needs two tickets.

In section 5.2.2 we have already calculated the number of tickets issued on average each second including delegation tickets for all machines involved. Thus we have seen that with full scale delegation, the Kerberos system has no problem handling all delegation in the data center. If the delegation assumptions are lowered, the load on the KDC will be lighter.

The depth of the delegation tree has no influence on the amount of processing- or networking power needed to delegate further. The workload for delegating from level l_a to level l_{a+1} is the same as the delegation from level l_b to level l_{b+1}, both on the delegating and the verifying side of the delegation.

5.2.4 Performance of repeated access and long-running jobs

Repeated access to services in Kerberos while the ticket is still valid does not require any extra work for the TGS. By presenting a valid ticket, authentication is achieved.

In case of long jobs, tickets may time out and need to be extended. In such a scenario, the delegation tree needs to be reseeded with tickets from the root up, based on requests from the nodes whose tickets expire. In the worst case scenario all 10 000 tickets in a job expire at once. This equals the peak load of the data center, which the Kerberos system can easily handle within one second, as seen in section 5.2.2.

On long-running jobs the probability increases for a machine failure in one of the worker machines. When a machine or job crashes, the job needs to be relocated or restarted respectively. In case of a restart this does not require resources from Kerberos, as the ticket is valid on the machine that started the job. In case the crashed job held tickets for further access, the access tree needs to be rebuilt from the crashed job. In the case a machine fails and the job needs to be relocated, the entire tree rooted in the failed note may need to be replaced. From the point of view of the TGS this rebuilding process is no different than a ticket timeout scenario, and we therefore still have the same result that the TGS will have no problems handling the resulting workload.

5.3 Performance of PKI

In this chapter the performance of a PKI system under the written assumptions is analysed.

5.3.1 Performance of central CA

The CA is responsible for issuing identity certificates to all entities, thereby also confirming their identity, but there is no direct connection between the operation of the CA and the continuous operation of the data center.

The amount of storage and the resources needed for a CA in a PKI of our size is reasonable, and is left out of this analysis. The possible exception to this is the option of a revocation service and will be discussed later.

5.3.2 Performance of certificate validation

Validation of a X.509 certificate path is standardised in RFC3280 [31]. The run time of the algorithm is dominated by the signature verification timings given in appendix A and repeated in table 5.2. The security of PKI is dependent on certificate validation routines. The theoretical maximum number of validations possible per second depends on the signature methods chosen, but is for certificates signed with RSA-1024 10 307 certificates. This is approximately 27% fewer than the numbers taken from the Crypto++ library benchmark as shown in table 2.2, which gives 14 285 verifications of RSA-1024 signatures per second. In reality the number would be even lower because of needed network communication and other processing tasks, and programs running in a PKI environment need to be aware of the possibility of high latency in the authentication process and not use too aggressive timeouts.

OpenSSL	sign (ms)	verify(ms)	sign/s	verify/s	20 000 verify +10 000 sign
rsa 512 bits	0.483	0.036	2 071.1	27 847.5	5.55s
rsa 1024 bits	2.208	0.097	452.8	10 307.2	24.02s
rsa 2048 bits	11.909	298	84.0	3 353.3	357s
rsa 4096 bits	72.899	987	13.7	1 013.2	748s
	sign (ms)	verify (ms)	sign/s	verify/s	20 000 verify +10 000 sign
dsa 512 bits	0.374	0.428	2 670.4	2 336.5	12.3s
dsa 1024 bits	0.966	1.148	1 034.9	871.1	32.62s
dsa 2048 bits	2.954	3.534	338.6	283.0	100s
Crypto++ 5.5	sign (ms)	verify (ms)	sign/s	verify/s	20 000 verify +10 000 sign
rsa 1024 bits	1.42	0.07	704.2	14 285.7	15.6s
rsa 2048 bits	5.95	0.15	168.1	6 666.7	62.5s
dsa 1024 bits	0.47	0.52	2 127.7	1 923.1	15.1s

Table 5.2: Timings of OpenSSL cryptographic operations
The base timings for OpenSSL are found in appendix A. The base timings for Crypto++ 5.5 is found in [Dai08].

5.3.3 The need for two-way authentication

Unlike in Kerberos, where the ticket is only readable by the correct recipient, in a PKI situation, authentication on both sides is necessary to avoid man-in-the-middle attacks [10]. For reverse authentication, a method can be used as outlined in section 3.3.4.

5.3.4 Considerations for crypto system selection

The choice of key used in the different public key scenarios must largely depend on the intended use of the keys. An RSA key enables both encryption and signatures, whereas a DSA key only enables signatures. If confidentiality is needed under DSA, a method such as DSA-secured Diffie-Hellman can be used, where DSA is used in DH to secure the integrity of the exchange from man-in-the-middle attacks.

Pre-generation of keys is possible in both crypto systems, but as RSA keys require more processing power to generate, the speed benefit will probably be greater if RSA keys are pre-generated than if DSA keys are pre-generated. The total processor load will in all cases be the same, whether pre-generated keys are used or not, so the total power consumption in the data center will be the same.

5.3.5 Needed key length

The BSI in Germany [9] has recommended key lengths to be at least 1024 bit long, both for DSA and RSA for security until 2007, and NIST has specified

that 1024 bit keys can be used until 2010 [4], as discussed in section 2.8.

For a temporary certificate it may be that a key length even shorter than 1024 is good enough to use, as the key will be invalid after a short period of time. However, it may also be that a temporary key might enable the resurrection of other temporary certificate, or that a key is used for other purposes or lengths of time than intended. The analysis will therefore not consider any public keys in either DSA or RSA shorter than 1024 bit.

5.3.6 Average and peak computational load of each machine

In order to authenticate a client, the server needs to verify the signature on the client certificate, and possibly verify other certificates in the chain. The server also needs to issue a random challenge to the client and verify the signature on the challenge. In addition, the server needs to sign a random challenge from the client in order to complete the two-way authentication. The resulting workload for the server is to verify two signatures and generate one signature for each authentication.

We use the same approach as for Kerberos to estimate the total number of individual service requests in the PKI environment. We do not, however, need any extra requests for delegation tokens, which gives the upper bound on service requests at 1 000 000 000 per day and the average number of service requests per day at 100 000 000.

As there is no central authentication server in PKI, we can average the number of individual hits on each machine by dividing the services evenly in the data center, leaving each machine with a number of requests per day of 10 000 in the worst case and 1000 requests per day in the average case.

If the requests are evenly distributed over the day, the maximum number of requests per day, 10 000, implies each machine has 8,6 seconds to handle each one, and the average number of requests per day, 1 000, gives each machine 86,4 seconds per request. From tables 5.2 and 2.2 we see that both these data center loads are able to be handled regardless of certificate key choice.

The peak load in the system, when 10 000 machines require authentication to one machine at the same time, for example a file server, cannot be handled without causing very high latency. We see that both Crypto++ and OpenSSL can verify all 10 000 certificates in one second if a small RSA modulus is chosen as the key base. However, none of the cryptographic libraries are able to handle the peak load in under 15 seconds given a key base of recommended strength.

This calculation also takes into consideration the situation where all 10 000 machines are delegated to directly by the user. As the user's certificate will be the same for all 10 000 presented certificate chains, this certificate only needs to be validated once.

Key type	lowest seen (ms)	highest seen (ms)
RSA 1024	46	125
RSA 2048	600	1170
DSA 1024	8	12
DSA 2048	16	19

Table 5.3: Informal measurements of key generation time
Measured on an Intel Core2 2.33GHz CPU with OpenSSL 0.9.8g

5.3.7 Performance of certificate issuing for delegation

In the X.509 proxy certificate example, each delegation requires a generation of a public/private key pair. Timings for such generation are informally measured and listed in table 5.3. The numbers for DSA are based on pre-generated parameter sets, a process only needed once.

If keys are generated by the delegates instead of generated centrally, key generation can be done in parallel, as described in section 3.3.3. The CA-issued certificate or the delegation certificate of each intermediary principal will create a new CSR, and the delegating principal will then sign the certificate. This eliminates the problem of key transfer that would occur if the secret keys were generated by the delegator, as the secret key is never transferred over the network, and it greatly reduces the strain on delegating principals from generating keys. If a solution with an attribute certificate is chosen, the key generation time is zero, but at the cost of always using the entity personal certificate for attribute certificate verification by linking the identity in the attribute certificate to the verifiable identity of the public-key certificate.

The time needed to issue a certificate from a CSR is again dominated by the public-key signing operation from table 5.2, and is compared to central issuing in table 5.4. As the authentication is performed before the delegation certificate is to be issued, we disregard the signature verification on the CSR, as the integrity of the CSR in all cases will hinge on the integrity of the established session.

We see that it takes 1,47 seconds in total to issue 1000 certificates with a 1024 bit key generated by the delegate and issued using a CSR if the issuer signs using a DSA-1024 key, whilst issuer-generation of the same 1000 certificates would have taken 10 seconds.

We see from the table that, so long as distributed key generation is used, the type of key in the certificate does not affect the speed of the issuing process. Thus the key in a certificate may be chosen based on the desired properties of a key, as described in section 5.3.4, rather than by issuing speed considerations alone. Also, as seen in section 5.3.6, the choice of key directly affects authentication latency in peak situations, and needs to be taken into consideration when the choice of key type is made.

5.3.8 Effects of the delegation path length

Contrary to Kerberos, a validation of a certificate for authentication involves the verification of all certificates in the certificate chain, not only the certificate

Issued certificate key / Issuer key	Any key in a CSR, or keyless attribute certificate	Equal to, issuer key, keys all generated by issuer
RSA 1024	2.21s	100 s
RSA 2048	11,91s	16 min
DSA 1024	0,97s	10 s
DSA 2048	2,95s	18 s

Table 5.4: Comparison of computation time needed on the issuer side to issue 1000 certificates
with different procedures and different issuer-keys. Timings are based on OpenSSL from table 5.2, and on key generation time from table 5.3.

of the authenticating node. Therefore, the level of a delegation in PKI influences greatly the computational cost of certificate chain verification.

As described in section 3.3.9, the certificate chain grows about 1KB for each level in the chain. This means that on level 10, each authentication requires transfer of 10 KB of data in addition to the reverse authentication, 10 times as much as a Kerberos ticket, and verification of all certificates in the chain is necessary for successful authentication. This may be mitigated somewhat by caching certificates, but the effects of caching is left out of this calculation because of the short lifetime of delegation keys.

With the help offigure 3.5, we can calculate the total amount of data transferred in the data center in a PKI delegation scenario using proxy certificates. For each level in the delegation, the delegate needs to acquire the certificate of all the previous nodes in the tree, and for all nodes except the delegator it needs also a delegation certificate. In addition to this, the level i delegate needs to send its own certificate and a delegation CSR to the level $i-1$ delegator. The total amount of data transferred in the scheme with CSRs from clients in a full delegation t-tree, disregarding the relatively small signature as included in the signature size variance, can thus be calculated as $2c\sum_{i=1}^{n}(i+1)\cdot t^i$, where c is the size of a certificate and a CSR, and n is the level of the last delegates.

The total amount of transferred data for two levels of delegation in a 10-tree, a tree where each machine delegates to 10 other machines, with 1KB certificates can be written as $2c\,((2\cdot 10)+(3\cdot 100)) = 640KB$. For three levels of delegation in a 10-tree, a job that has a total of 1110 machines, the amount of data transferred totals to $8640KB$.

For Kerberos, each machine needs two tickets for delegation. The total for a 1110 machine Kerberos job amounts to 2220KB of data assuming messages are 1KB. We see that the average PKI job with 10-tree delegation structure needs only 4MB extra data transfer in total for authentication and delegation.

Delegation depth also influences the verification times of authentication. Veri-

fication of the certificates needs to be done in all steps of the operation. From table 2.2 we can see that this operation is already substantially more costly than direct secret-key decrypting of a Kerberos token. For a level i delegation, the end point after the ith delegate needs to verify the certificate of the principal, | CA | Principal |, and then in sequence all the signed certificates from | Principal | 1 for Principal | to | $i-1$ | i for Principal | plus | CA | i |, totalling $i+1$ signatures.

For a 10-level delegation, the validation of the certificate chain will therefore take 11 times as long as the direct validation of the principal certificate. For the peak performance scenario, this means that if all 10 000 machines are in a tree delegation configuration with more than one level, the certificate validation time will be longer than the calculation in section 5.3.6.

As a contrast to Kerberos, we see that the delegation levels in PKI has great influence on the total workload in the data center.

A cryptographic technique that could increase the performance of PKI is the use of cascading signatures. A cascading signature is a signature that does not grow even though the signed material grows, as the new signature is combined with the previous signature. There are different technologies for implementing cascading signatures. One example of use of cascaded signatures is described in [50], where standard RSA keys are used, and an entire delegation chain can be verified by one short signature of constant size.

Chapter 6

Discussion

In this chapter we summarise the findings of the analysis and discuss the different uses of the discovered solutions, their strengths and their weaknesses.

6.1 Choice of subjects, and key properties of the systems

Kerberos and PKI are both widely deployed in large scale authentication systems. Kerberos is in use in large scale Windows domains, and PKI is in use in large scale grid computing systems. They have both been proved functional for big systems, and are therefore natural candidates for a first analysis of scalability in a multi-thousand computer data center as defined in this thesis.

The security in both systems is based on cryptography and on the chosen keys and algorithms. In Kerberos symmetric cryptography is used. Symmetric cryptography is very fast compared to public key operations, and reasonably easy to implement. It provides good security on relatively short key lengths compared to RSA. There are a number of symmetric key ciphers in existence. We have used the American standard cryptography, the AES-algorithm for calculations on the analysis, but Kerberos can run on any symmetric key cipher. Our calculations show that with AES as the cipher, the Kerberos server can easily handle the defined peak load of the system. However, if the peak load was defined to be 10 times as high, the system would not be able to handle it within the given time limit, as mentioned. The calculations also show that the network speed is faster than the crypto-speed, and therefore that the speed of the system is limited to the cryptographic algorithm.

The major disadvantage of symmetric ciphers is the key exchange problem. Any two partners in a conversation needs a shared secret key, and this key cannot be transferred openly over the network. Therefore the initial key agreement is a problem in a symmetric key environment. In addition, we need to take into account the massive number of keys existing in the symmetric key environment. In a normal mesh network the number of keys grows exponentially, but in Kerberos we are limited to linear key growth as the permanent secret

keys, as opposed to the generated session keys, are only shared with the KDC. Our calculations show that the KDC is able to keep all the keys in RAM, and therefore looking up keys of the individual principals as they are needed is not a bottleneck in the system.

When the tickets are issued, the security of the system is complete as long as the keys of the services are kept secret. The ticket is not decryptable or falsifiable by any principal, and therefore acts as a proof of authentication by the KDC. Moreover, as the information about the principal is embedded in the ticket, the KDC has guaranteed the identity of the principal to the service. As the ticket also includes a symmetric session key, the eavesdropping of a ticket and use by a malicious principal will result in communication from the service that is unreadable to the malicious principal because he does not know the session key.

In the PKI system the security of the system is based on the certified link between the principals and their public keys. The first of these links are certified by the CA, and the rest of the links in the certificate chain are verified by the previous certificates in the chain. A certificate stating that a principal has a certain key can be verified with the public key belonging to the certificate issuer.

As with symmetric cryptography, there are many algorithms for public key cryptography. We have chosen to do calculations on RSA and DSA. The RSA algorithm can be used both for signatures and encryption, whereas the DSA algorithm can only be used for signatures. The advantage of DSA over RSA is that DSA keys are substantially cheaper to generate, and also cheaper in total time for signature generation and verifications when the key size is larger than 1024 bits. In RSA, the signing and signature verification times are determined by the sizes of the private and public exponents. The choice of the public exponent is free, and a low number can chosen low to give a low encryption and signature verification time, but as described the private exponent needs to be larger than a certain root of the RSA modulus in order to be secure, and therefore the cost of signing cannot be reduced further than this root allows. In addition, problems may also occur if the public exponent is chosen too low. In both cases, the public crypto techniques are so costly that they need to be used only as a basis for agreeing on a symmetric shared key for further data exchange.

The identity verification in the certificate lies in the fact that the owner of the public key is the only one with knowledge of the private counterpart of the key. Upon presentation of the certificate, the verifier needs to check that the signature on the certificate is correct, and also that the presenter knows the private part of the key stated in the certificate. If this can be proved, the identity of the owner of the private key is verified by the certified link between the identity and the public key.

Both systems also have very versatile delegation possibilities, and therefore fit well with the desired properties of an authentication system. PKI is clearly the most versatile of the two, as X.509 is prepared for extensions and therefore can contain extensive information in an already defined way. Such extensions may include properties stating that resource A is only accessible on Sundays or that the certificate is not valid on Mondays. We have not looked at the possibility of adding extension information in the Kerberos protocol. PKI supports the possibility of doing traceable delegation by using the personal certificates in the

chain, and untraceable delegation by using anonymous certificates. Kerberos does not have these possibilities, but Kerberos can choose whether to issue proxy or delegation tickets to control whether or not the tickets can be used for further delegation.

6.2 Administration

Ease of administration is a key factor in a large scale data center. In both systems analysed we have described concepts that makes it possible to automate all the basic steps of inserting new machines, users and services in the data center. A key factor in this is that all out of band initial keys may be stored on the installation medium, e.g. the CA certificate, and the physical properties of the machine itself can be used to safely agree on initial keys.

We have described methods for both Kerberos and PKI that enable automatic initialisation of new machines in the data center. By using serial numbers on the hardware as a trust anchor, the only out-of-band signalling needed is that the serial numbers of the machines are given to the central service. This means that the initialisations can be done with a common installation media. This method might not be completely secure if serial numbers are not large enough or random enough. It is also not completely secure for initialisation of the same hardware more than once. However, we feel that the method establishes very high security, when weighed against the needs for automation in a large data center.

By using the serial number only as a basis for establishing the secret machine key, we make sure that it is not possible to derive the secret key directly from the serial number. This is important as an attacker may at some point see the serial number by visual inspection of the machine or by uploading a service that reads the serial number out from the hardware.

We have described two fundamental ways of inserting new users in the data center: the password-entry via secure channels; and the issuing of a certificate via smart-cards. Both these methods are in use in various solutions today, and as our number of users is not a large number compared to any network in use today, enrolling new users can be solved using standard methods in Kerberos and PKI.

To insert new services and service instances in the data center automatically we have described a method utilising the concept of a service owner. The Kerberos insertion process involves the service owner registering the instance with the Kerberos service under a unique name to receive a Kerberos key for the given service, and then proceeding to upload the service to a machine. This ensures that service names are unique and unforgeable. In PKI the same situation is achieved by issuing certificates to the services with the service owner certificate, as done in grid computing systems. This likewise ensures that a service can not be given a certificate for a false name.

6.3 Error handling and fault tolerance

For both systems, a problem in the authentication algorithm can lead to unauthorised access or false denial of access. This is the same for all access control systems. For further discussion we refer to [25].

In case of central server malfunction, the PKI system would still be able to operate. In case a solution is chosen that operates without revocation lists (meaning it operates with short certificate lifetimes) the system would not necessarily know if the CA was down. In the Kerberos situation, the system could only continue to function in the state it was when the central system fell down. As discussed, valid tickets would still lead to successful authentication and operation, but no tickets could be renewed or issued.

Introducing failover servers is easy both for PKI and Kerberos. As neither system relies on state stored on the central server, each server could function separately. Two different CAs also do not need to share a key, as long as all entities in the system are given all the CA certificates out of band. There is also a possibility of introducing a second CA while the data center is running, if the second CA's certificate is issued by the first CA. The certificate chain would be one step longer, but as the second CA is common to all parties, this certificate would only need to be validated once.

In a solution with several Kerberos servers, all would need to have access to the database of shared secrets. This means that in order to avoid a central point of failure, the database would have to exist once for each failover Kerberos server. This will also require technology for database synchronisation in order to make sure that all servers are accessible to all entities. It is especially important that the synchronisation solution enforces the integrity of the database, as the database as a whole is still a central point of failure even with replication in place.

6.4 Trust

As the system is described, the trust in the system is only between the individual entities in the system and the CA or the KDC. In addition the trusted service on the machine is trusted by the applications, but this trust is not conceptually different than the trust in the machine itself, and so compromise of the machine by compromising the operating system or compromising the trusted service is equivalent.

By designing the system this way, the assumptions on the system are as weak as possible. For an application to run on a machine, the machine and the operating system are implicitly trusted, as the operating system controls all the machines resources. As all entities in the system have unique credentials in the form of a trust relationship with the Kerberos server or in the form of a certificate, there is no difference in the operation of the system depending on the trust of any entity.

Even though the system operates with mutually sceptical entities, there is a need to establish trust in groups of services. A hostile service cannot be inserted into

the group of known file service instances, for example, and a mechanism for this needs to be implemented in the system. For PKI this is achieved through the use of a service master issuing certificates to all instances of a service. Only services providing a certificate issued by the service master are trusted as legitimate instances. Via the trusted service, a master service can establish new instances of its service in the data center dynamically. The user of a given service must trust the master server. It would be a reasonable assumption that important services such as fileservers have a master on a very well secured machine.

6.5 Performance

The premises for the analysis state a 100 000 machine data center with 1000 users. Each job may run on as many as 10 000 machines. We use assumptions on the behaviour of users and the number of possible machines per job to achieve the average number of connections per day, ending up with 100 000 000 individual requests. The correctness of this number is debatable, but still forms a good basis for calculations. A 100 000 machine data center does not exist, a point that is repeated in this thesis. Any estimation on a number of connections is therefore only vague. Real numbers of connections in data centers are closely guarded secrets, and getting the real number of connections from even a comparable data center is not an option for this thesis.

We have seen in our analysis of the performance of the two technologies that Kerberos is, even with one single Kerberos server, able to handle both the average and peak loads expected in our data center. We have also seen that the PKI system was able to handle the average load. Due to the distributed nature of PKI, each machine had roughly 8,6 seconds per request to deal with the validation of certificates and the two-way authentication algorithm. On the other hand the PKI system did not have the resources to deal with our defined peak load of an entire job authenticating to one single machine. This was because the public key cryptography was not able to verify that number of signatures and do the two-way authentication procedure in the course of one second. In fact, one PKI node needs 15 seconds to handle 10 000 authentications, and that is when the delegation chains of all the authenticating principals was only two levels long. In case the delegation tree was differently configured, the time needed for authentication would rise.

The impact of increasing the number of users ten fold is bearable in the average situation on both PKI and Kerberos. This would mean that the average number of Kerberos tickets issued each second rises to today's peak load as described in section 5.2.2, - which the KDC is able to handle based on our analysis, and that the mean time for one ticket goes down from 8 seconds to 0,8 seconds in the average PKI scenario as seen in section 5.3.6. In the same sections, we see that both in the Kerberos and the PKI environment, given the correctness of our assumptions, the data center cannot handle a peak load 10 times as high without developing severe bottlenecks.

With regards to the delegation performance, the analysis shows that while there is no apparently needed pattern to follow in Kerberos delegation, with the exception of taking care in the ticket expiration time selection, there is a possibility of

implementing delegation wrongly in the PKI scenario. Specifically, the analysis shows that delegation trees in PKI should not be too deep, and that intermediate keys needs to be generated by the delegates and not the delegating party, unless keyless attribute certificates are used.

The analysis shows that the latency in PKI is relatively high compared to Kerberos, and a longer certificate chain results in longer latency. The latency in both technologies is nevertheless present, even though we have disregarded the network performance in our analysis. As shown in the theory sections and in the appendices, the speed of even the quickest symmetric key cryptographic algorithms are substantially slower than 10GBPS network speed. It is to be assumed that any data center with 100 000 machines is equipped with such a networking system. The latency of the network is also disregarded, as we can see that the number of packets in each step in both Kerberos and PKI is in the order of tens. The impact of network latency in our systems is therefore dominated by the speed of the cryptographic operations in each step.

Chapter 7

Conclusion

We have shown in our analysis that both PKI and Kerberos are alternative authentication mechanisms for a data center with 100 000 machines.

From the administration analysis we have seen that potential solutions exist in both systems for automatically introducing new machines for the first time in the data center. The method we have proposed makes it possible to ensure a reasonable amount of initial channel security in this process. We have examined the solution where a trusted service exists on each machine and is responsible for the machine key as well as functioning as the administrative interface on the machine. We have also seen that there are methods for both systems to introduce new users in the system, and that these procedures are well tested and employed in existing systems.

In Kerberos we have proposed a potential solution for introducing new services based on the use of a master service or service owner. This solution ensures unique names for all services in the data center, and also ensures that each process is given a key over a secure channel. In PKI we have examined a process based on the same idea with a service owner, but utilising ideas from grid systems to examine the issuer certificate to authenticate processes. We have also examined how this solution can be used to enable services to span several machines. For PKI this does not imply common shared keys, whilst for Kerberos a multiple-instance scenario might require shared keys between all services unless an additional indirection layer is added.

From the fault handling analysis we have seen that Kerberos is the system that requires the least amount of computing power from each node in the system, but that as Kerberos relies heavily on a central point, special care needs to be taken to ensure security and functionality in failure situations. PKI systems do not rely operationally on the central CA, and are therefore by design capable of handling crash-faults as we have described them. This situation will change if a central revocation system is chosen, but this is not examined in detail in this thesis.

From the performance analysis we see that the Kerberos system is capable of dealing with both the average load and the defined peak load of the data center with only one single TGS. The PKI system is distributed by nature, and under

average load, the machines in the PKI system encounter no problems with the traffic. Under our defined peak load, however, the PKI system would introduce a significant latency in the system. This is because a single machine is not able to run through the authentication algorithm 10 000 times in less than 15 seconds when running with a recommended key length. Applications may need to be specially configured to handle this latency.

Finally, we have shown that in our hypothetical data center, both systems are able to handle delegation according to the system standards. However, there are some restrictions on how delegation for PKI can be implemented in order to meet the performance requirements of the system. These restrictions come as a result of the relatively time-consuming task of key generation. It is essential in a PKI delegation scenario that each delegate generates a CSR and thereby also its own keys. This means that each node also needs the cryptographic capabilities these operations require.

Chapter 8

Future work

As this field of study is quite new, there are many paths for future work. Possibly the most important work that needs to be done is to construct and build a simulation system for a large data center to find suitable parameters for the systems we have looked at in this report.

Further work also needs to be done in constructing mechanisms for authorisation in the data center. Both Kerberos and PKI have the possibility to act as authorisation mechanisms in themselves, but this has not been considered in this thesis. Without an authorisation service, the authentication service does not make a complete access control system.

As shown in the analysis, the peak load of the system as we define it results in 15 seconds latency for the PKI system. It would be interesting to look at how this affects a real system, and what techniques could be used to reduce the time needed for the peak load handling.

During the course of writing this thesis, many interesting aspects of delegation were considered. In particular, the mentioned cascading signature scheme discussed in section 5.3.8 would be interesting to study further.

We have not looked at how multiple data centers can cooperate under Kerberos or PKI. It would be interesting to study how this may be implemented using either basic replication, Kerberos's realm-system or using ideas from grid computing. Such a study would preferably also look at solutions for service naming and discovery.

The effect of access control on the development environment is also an issue it would be interesting to investigate. It should be as transparent to the developers as possible, while still maintaining the highest possible security.

Virtualisation technologies are starting to gain a strong foothold in data centers. It would be interesting to see how a solution for application deployment based on virtualisation may solve some of the deployment problems we have looked at in this thesis.

References

[1] IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. *IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005)*, 2006.

[2] Federal Information Processing Standards Publication 197. Announcing the advanced encryption standard (AES). 2001.

[3] H. Baier and M. Ruppert. Interoperable and Flexible Digital Signatures for E-Government and E-Commerce. 2004.

[4] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. *Recommendation for Key Management – Part 1: General (Revised).* NIST National Institute of Standards and Technology, NIST special publication 800-57 edition, March 2007.

[5] Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, and Kami Vaniea. Lessons learned from the deployment of a smartphone-based access-control system. In *SOUPS '07: Proceedings of the 3rd Symposium on Usable Privacy and Security*, pages 64–75, July 2007.

[6] Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Information Security: 8th International Conference, ISC 2005*, volume 3650 of *Lecture Notes in Computer Science*, pages 431–445, September 2005.

[7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. *Advances in Cryptology-Eurocrypt 2000*, 2000.

[8] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *Lecture Notes in Computer Science*, 1592, 1999.

[9] BSI. *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen).* Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, Februar 2007.

[10] J. Buchmann. *Einführung in die Kryptographie.* Springer, 3. erweiterte edition, 2003.

[11] Miguel Castro, Rodrigo Rodrigues, and Barbara Liskov. Using abstraction to improve fault tolerance. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 27, Washington, DC, USA, 2001. IEEE Computer Society.

[12] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.1. RFC 4346, Internet Engineering Task Force, April 2006.

[13] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[14] Yun Ding, Patrick Horster, and Holger Petersen. A new approach for delegation using hierarchical delegation tokens. In *Proceedings of the IFIP TC6/TC11 international conference on Communications and multimedia security II*, pages 128–143, London, UK, UK, 1996. Chapman & Hall, Ltd.

[15] Maryia Drahavets. Flexible anbindung von smartcards an eine sicherheitsinfrastruktur. Master's thesis, Technische Universität Darmstadt, August 2006.

[16] S. Farrell and R. Housley. An internet attribute certificate profile for authorization, 2002.

[17] David Ferraiolo and Richard Kuhn. Role-based access control. In *Proceedings of 15th NIST-NCSC National Computer Security Conference*, pages 554 – 563, 1992.

[18] E.A.U.P.K. FIPS, Cryptography. Entity Authentication Using Public Key Cryptography. *Feb*, 18:1–52, 1997.

[19] I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1998.

[20] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum, June*, 22:2002, 2002.

[21] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. *Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, 1998.

[22] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[23] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, et al. The Open Grid Services Architecture, version 1.5. *Global Grid Forum. Retrieved February*, 17:2007, 2006.

[24] Pierre-Alain Fouque, Sébastien Kunz-Jacques, Gwenaëlle Martinet, Frédéric Muller, and Frédéric Valette. Power attack on small RSA public exponent. *Cryptographic Hardware and Embedded Systems - CHES 2006*, 4249/2006:339–353, 2006.

[25] Håvard Husevåg Garnes and Petter Wedum. Innbruddstesting på prosesskontrollsystemer for oljeplattform. *Fordypningsprosjekt i informasjonssikkerhet, ITEM, NTNU*, 2007.

[26] M. Gasser and E. McDermott. An Architecture for Practical Delegation in a Distributed System. *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, pages 20–30, 1990.

[27] Frédéric Gittler and Anne C. Hopkins. The DCE Security Service. *Hewlett-Packard Journal*, December, 1995.

[28] A.H. Harbitter and D.A. Menasce. Performance of public-key-enabled kerberos authentication in large networks. *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 170–183, 2001.

[29] R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed environment. *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 3–14, 3-6 May 1998.

[30] M.D. Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.

[31] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, 2002.

[32] J. Kohl and C. Neuman. The kerberos network authentication service (v5), 1993.

[33] John T. Kohl, B Clifford Neumann, and Theodore T. Ts'o. The evolution of the kerberos authentication service. In *Proceedings of the Spring 1991 EurOpen Conference*, 1991.

[34] Fang-Chun Kuo, Hannes Tschofenig, Fabian Meyer, and Xiaoming Fu. Comparison studies between pre-shared key and public key exchange mechanisms for transport layer security (tls). *Technische Berichte des Instituts für Informatik an der Georg-August-Universität Göttingen*, IFI–TB–2006–001, 2006.

[35] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[36] Alexander May. Skript zur vorlesung public key kryptanalyse. Technical report, TU-Darmstadt, 2006.

[37] S. Micali. Enhanced Certificate Revocation System. 1995.

[38] S. Micali. Efficient certificate revocation. United States Patent and Trademark Office, US Patent 6,487,658, 2002.

[39] J.A. Montenegro and F. Moya. A Practical Approach of X. 509 Attribute Certificate Framework as Support to Obtain Privilege Delegation. *Public Key Infrastructure: First European PKI Workshop: Research and Applications, EuroPKI 2004, Samos Island, Greece, June 25-26, 2004: Proceedings*, 2004.

[40] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.

[41] B.C. Neuman, TL Casavant, and M. Singhal. Scale in distributed systems. *Readings in Distributed Computing Systems*, pages 463–489, 1994.

[42] Daniel Nussbaum and Anant Agarwal. Scalability of parallel machines. *Commun. ACM*, 34(3):57–61, 1991.

[43] Deparment of Defence. *Trusted Computer System Evaluation Criteria (Orange Book)*. Department of Defence, 1985.

[44] National Institute of Standards and Technology (NIST). Fips publication 180-2: Secure hash standard. 2002.

[45] M. Rahnema. Overview of the gsm system and protocol architecture. *Communications Magazine, IEEE*, 31(4):92–100, Apr 1993.

[46] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[47] Howard Jay Siegel. *Interconnection networks for large-scale parallel processing: theory and case studies (2nd ed.)*. McGraw-Hill, Inc., New York, NY, USA, 1990.

[48] M. Stamp. *Information security: principles and practice*. Wiley-Interscience, 2006.

[49] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202, Berkeley, CA, 1988. USENIX Association.

[50] R. Tamassia, D. Yao, and W.H. Winsborough. Role-based cascaded delegation. *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 146–155, 2004.

[51] K. Vedder. GSM: Security, Services, and the SIM. *State of the Art in Applied Cryptography: Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 3-6, 1997: Revised Lectures*, 1998.

[52] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X. 509 Proxy Certificates for Dynamic Delegation. *3rd Annual PKI R&D Workshop*, 14, 2004.

[53] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid services. *12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings.*, pages 48–57, 2003.

[54] Gregory White and Udo Pooch. Problems with dce security services. *SIGCOMM Comput. Commun. Rev.*, 25(5):5–12, 1995.

[55] Ronny Windvik, Martin Gilje Jaatun, and Geir Hallingstad. Security Mechanisms In Windows NT. FFI Notat, March 2001. 2001/01395.

[56] ITU-T Recommendation X.509. Information technology – open systems interconnection – the directory: Public-key and attribute certificate frameworks. *SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY*, 2005.

[57] E.D. Zwicky, S. Cooper, and D.B. Chapman. *Building Internet Firewalls*. O'Reilly, second edition, 2000.

[58] André Årnes. Public key certificate revocation schemes. Master's thesis, Department of Telematics, Norwegian University of Science and Technology, 2000.

References from the WWW

[All08] The Globus Alliance. The globus alliance web site. http://www.globus.org/, Retrieved February 2008.

[Car08] David F. Carr. Baseline magazine, how google works. http://www.baselinemag.com/c/a/Projects-Networks-and-Storage/How-Google-Works-[1]/, Retrieved April 2008.

[Cou] PCI Security Standards Council. Home - pci security standards council. https://www.pcisecuritystandards.org/.

[Dai08] Wei Dai. Crypto++ 5.5 benchmarks. http://www.cryptopp.com/benchmarks.html, Retrieved January 2008.

[Gla08] Eric Glass. The ntlm authentication protocol and security support provider. http://davenport.sourceforge.net/ntlm.html, Retrieved February 2008.

[Gro08] The Open Group. Dce portal. http://www.opengroup.org/dce/, Retrieved March 2008.

[Kam08] Satoshi Kambayashi. The economist - sarbanes-oxley, five years under the thumb. http://www.economist.com/displaystory.cfm?story_id=9545905, Retrieved June 2008.

[Mic08a] Microsoft. Job details, software development engineer, job code: 219869. http://members.microsoft.com/careers/search/details.aspx?JobID=1441faf4-f9c8-4354-ab69-683d00257491, Retrieved April 2008.

[Mic08b] Microsoft Help and Support. How to force kerberos to use tcp instead of udp in windows server 2003, in windows xp, and in windows 2000. http://support.microsoft.com/kb/244474, Retrieved Feruary 2008.

[MIT08] MIT. Kerberos: The network authentication protocol. http://web.mit.edu/kerberos/, Retrieved February 2008.

[OV08] Alberto Ornaghi and Marco Valleri. Ettercap. http://ettercap.sourceforge.net/, Retrieved April 2008.

[Pro08] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. http://www.openssl.org/, Retrieved February 2008.

[Tec08] Microsoft TechNet. Single sign-on in windows 2000 networks. http://technet.microsoft.com/en-us/library/bb742456.aspx, Retrieved February 2008.

[Wik08] Wikipedia. Sneakernet. http://en.wikipedia.org/wiki/Sneakernet, Retrieved April 2008.

[WSOW08] Alex Weeks, Stephen Stafford, Joanna Oja, and Lars Wirzenius. The linux system administrator's guide chapter 10.1. logins via terminals. http://tldp.org/LDP/sag/html/login-via-terminal.html, Retrieved Feruary 2008.

Appendix A

Results of cryptographic benchmarks

This section shows the results of benchmarks on my own computer, a Thinkpad T60p with an Intel(R) Core(TM)2 CPU T7600 @ 2.33GHz processor.

A.1 OpenSSL benchmark

```
$ openssl speed
OpenSSL 0.9.8g 19 Oct 2007
built on: Tue Apr 22 07:35:00 UTC 2008
options:bn(64,32) md2(int) rc4(idx,int)
        des(ptr,risc1,16,long) aes(partial)
        blowfish(idx)
compiler: gcc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS
              -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIAN
              -DTERMIO -O3 -march=i686 -Wa,--noexecstack -g
              -Wall -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL_IA32_SSE2
              -DSHA1_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM
available timing options: TIMES TIMEB HZ=100 [sysconf value]
timing function used: times
The 'numbers' are in 1000s of bytes per second processed.
type            16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md2              2302.27k     4868.59k     6546.09k     7386.11k     7630.34k
mdc2                0.00         0.00         0.00         0.00         0.00
md4             32144.78k   107831.19k   295050.58k   518169.60k   666602.15k
md5             25800.85k    81993.32k   207122.52k   334497.45k   408322.05k
hmac(md5)       25650.26k    79665.47k   203191.64k   331165.70k   408573.27k
sha1            25964.21k    78333.50k   180622.85k   268824.23k   313636.18k
rmd160          19356.60k    51844.03k   103606.27k   138560.17k   153220.44k
rc4            255885.94k   276963.31k   286451.03k   262796.29k   210059.26k
des cbc         51881.92k    53734.31k    54303.66k    54388.74k    54504.11k
des ede3        19661.45k    20059.88k    20144.55k    20042.84k    20182.11k
```

```
idea cbc          0.00         0.00         0.00         0.00         0.00
seed cbc          0.00         0.00         0.00         0.00         0.00
rc2 cbc       23026.47k    23897.02k    24053.33k    24057.51k    23737.69k
rc5-32/12 cbc     0.00         0.00         0.00         0.00         0.00
blowfish cbc  84697.74k    89095.19k    90426.03k    90699.09k    90843.82k
cast cbc      44467.01k    46123.33k    47055.02k    47216.64k    47300.61k
aes-128 cbc   87904.22k   113397.57k   120894.12k   125095.94k   126812.16k
aes-192 cbc   78275.38k    97564.12k   104193.02k   106030.08k   106779.99k
aes-256 cbc   70314.73k    85802.86k    90403.93k    91564.37k    92091.73k
camellia-128 cbc  0.00         0.00         0.00         0.00         0.00
camellia-192 cbc  0.00         0.00         0.00         0.00         0.00
camellia-256 cbc  0.00         0.00         0.00         0.00         0.00
sha256        12218.43k    27663.73k    46293.79k    57797.09k    61966.04k
sha512        10182.86k    40813.81k    67631.09k    98004.65k   113274.26k
aes-128 ige   99883.91k   107984.98k   109842.69k   110211.07k   109840.84k
aes-192 ige   87238.21k    93345.79k    95264.85k    95782.23k    95234.73k
aes-256 ige   77104.20k    82242.94k    83961.06k    84280.66k    83602.09k
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000483s 0.000036s   2071.1  27847.5
rsa 1024 bits 0.002208s 0.000097s    452.8  10307.2
rsa 2048 bits 0.011909s 0.000298s     84.0   3353.3
rsa 4096 bits 0.072899s 0.000987s     13.7   1013.2
                  sign    verify    sign/s verify/s
dsa  512 bits 0.000374s 0.000428s   2670.4   2336.5
dsa 1024 bits 0.000966s 0.001148s   1034.9    871.1
dsa 2048 bits 0.002954s 0.003534s    338.6    283.0
```

A.2 Crypto++ benchmark

/cryptopp552$ ./cryptest.exe b

Algorithm	MiB/Second	Microseconds to setup Key and IV
VMAC(AES)-64	1914	2.394
VMAC(AES)-128	994	2.774
HMAC(SHA-1)	175	0.550
Two-Track-MAC	139	0.037
CBC-MAC/AES	107	0.249
DMAC/AES	108	1.299
CRC-32	376	
Adler-32	595	
MD5	361	
SHA-1	183	
SHA-256	82	
SHA-512	125	
Tiger	278	
Whirlpool	73	
RIPEMD-160	113	
RIPEMD-320	114	
RIPEMD-128	202	
RIPEMD-256	186	

Algorithm	MiB/Second	Microseconds to setup Key and IV
Panama-LE	1089	1.362
Panama-BE	288	3.551
Salsa20	522	0.249
Salsa20/12	833	0.316
Salsa20/8	1164	0.324
Sosemanuk	949	0.989
MARC4	164	2.898
SEAL-3.0-BE	403	55.881
SEAL-3.0-LE	420	55.881
WAKE-OFB-BE	246	2.698
WAKE-OFB-LE	256	2.690
AES/ECB (128-bit key)	121	0.199
AES/ECB (192-bit key)	105	0.215
AES/ECB (256-bit key)	96	0.240
AES/CTR (128-bit key)	120	0.532
AES/OFB (128-bit key)	107	0.470
AES/CFB (128-bit key)	77	0.643
AES/CBC (128-bit key)	108	0.352

Algorithm	MiB/Second	Microseconds to setup Key and IV
Camellia/ECB (128-bit key)	72	0.211
Camellia/ECB (256-bit key)	58	0.260
Twofish	65	4.209
Serpent	42	0.636
CAST-256	40	1.290
RC6	84	1.843
MARS	76	1.981
SHACAL-2/ECB (128-bit key)	62	0.480
SHACAL-2/ECB (512-bit key)	62	0.492
DES	43	7.398
DES-XEX3	38	7.750
DES-EDE3	16	23.165
IDEA	50	0.179
RC5 (r=16)	101	1.401
Blowfish	63	63.477
TEA/ECB	36	0.177
XTEA/ECB	33	0.176
CAST-128	50	0.408
SKIPJACK	27	2.359

Operation	ms/operation
RSA 1024 Encryption	0.06
RSA 1024 Decryption	1.20
LUC 1024 Encryption	0.07
LUC 1024 Decryption	1.92
DLIES 1024 Encryption	0.67
DLIES 1024 Encryption with precomputation	1.14
DLIES 1024 Decryption	2.40
LUCELG 512 Encryption	0.47
LUCELG 512 Encryption with precomputation	0.47
LUCELG 512 Decryption	0.50
RSA 2048 Encryption	0.14
RSA 2048 Decryption	5.05
LUC 2048 Encryption	0.16
LUC 2048 Decryption	7.87
DLIES 2048 Encryption	3.25
DLIES 2048 Encryption with precomputation	3.61
DLIES 2048 Decryption	14.49
LUCELG 1024 Encryption	1.50
LUCELG 1024 Encryption with precomputation	1.49
LUCELG 1024 Decryption	1.34
RSA 1024 Signature	1.25
RSA 1024 Verification	0.06
RW 1024 Signature	1.72
RW 1024 Verification	0.04
LUC 1024 Signature	1.91
LUC 1024 Verification	0.07
NR 1024 Signature	0.35
NR 1024 Signature with precomputation	0.32
NR 1024 Verification	0.40
NR 1024 Verification with precomputation	0.49
DSA 1024 Signature	0.36
DSA 1024 Signature with precomputation	0.33
DSA 1024 Verification	0.40
DSA 1024 Verification with precomputation	0.53
LUC-HMP 512 Signature	0.48
LUC-HMP 512 Signature with precomputation	0.47
LUC-HMP 512 Verification	0.49
LUC-HMP 512 Verification with precomputation	0.48
ESIGN 1023 Signature	0.19
ESIGN 1023 Verification	0.06
ESIGN 1536 Signature	0.33
ESIGN 1536 Verification	0.12
RSA 2048 Signature	5.05
RSA 2048 Verification	0.14

Operation	ms/operation
RW 2048 Signature	5.99
RW 2048 Verification	0.09
LUC 2048 Signature	7.94
LUC 2048 Verification	0.16
NR 2048 Signature	1.65
NR 2048 Signature with precomputation	0.74
NR 2048 Verification	1.85
NR 2048 Verification with precomputation	1.21
LUC-HMP 1024 Signature	1.50
LUC-HMP 1024 Signature with precomputation	1.50
LUC-HMP 1024 Verification	1.54
LUC-HMP 1024 Verification with precomputation	1.53
ESIGN 2046 Signature	0.41
ESIGN 2046 Verification	0.13
XTR-DH 171 Key-Pair Generation	0.65
XTR-DH 171 Key Agreement	1.30
XTR-DH 342 Key-Pair Generation	1.52
XTR-DH 342 Key Agreement	3.07
DH 1024 Key-Pair Generation	0.36
DH 1024 Key-Pair Generation with precomputation	0.59
DH 1024 Key Agreement	0.93
DH 2048 Key-Pair Generation	1.67
DH 2048 Key-Pair Generation with precomputation	1.86
DH 2048 Key Agreement	3.09
LUCDIF 512 Key-Pair Generation	0.25
LUCDIF 512 Key-Pair Generation with precomputation	0.25
LUCDIF 512 Key Agreement	0.49
LUCDIF 1024 Key-Pair Generation	0.76
LUCDIF 1024 Key-Pair Generation with precomputation	0.76
LUCDIF 1024 Key Agreement	1.33
MQV 1024 Key-Pair Generation	0.34
MQV 1024 Key-Pair Generation with precomputation	0.30
MQV 1024 Key Agreement	0.70
MQV 2048 Key-Pair Generation	1.61
MQV 2048 Key-Pair Generation with precomputation	0.72
MQV 2048 Key Agreement	3.12
ECIES over GF(p) 256 Encryption	4.76
ECIES over GF(p) 256 Encryption with precomputation	3.79
ECIES over GF(p) 256 Decryption	3.37
ECNR over GF(p) 256 Signature	2.42
ECNR over GF(p) 256 Signature with precomputation	1.93
ECNR over GF(p) 256 Verification	7.81
ECNR over GF(p) 256 Verification with precomputation	3.22
ECDHC over GF(p) 256 Key-Pair Generation	2.40
ECDHC over GF(p) 256 Key-Pair Generation with precomputation	1.89
ECDHC over GF(p) 256 Key Agreement	2.39

Operation	ms/operation
ECMQVC over GF(p) 256 Key-Pair Generation	2.39
ECMQVC over GF(p) 256 Key-Pair Generation with precomputation	1.90
ECMQVC over GF(p) 256 Key Agreement	7.94
ECIES over GF(2^n) 233 Encryption	18.36
ECIES over GF(2^n) 233 Encryption with precomputation	5.08
ECIES over GF(2^n) 233 Decryption	10.53
ECNR over GF(2^n) 233 Signature	9.26
ECNR over GF(2^n) 233 Signature with precomputation	2.60
ECNR over GF(2^n) 233 Verification	11.24
ECNR over GF(2^n) 233 Verification with precomputation	4.37
ECDHC over GF(2^n) 233 Key-Pair Generation	9.09
ECDHC over GF(2^n) 233 Key-Pair Generation with precomputation	2.54
ECDHC over GF(2^n) 233 Key Agreement	9.26
ECMQVC over GF(2^n) 233 Key-Pair Generation	9.17
ECMQVC over GF(2^n) 233 Key-Pair Generation with precomputation	2.53
ECMQVC over GF(2^n) 233 Key Agreement	11.48